3D Models for...

Géraldine Morin

Institut National Polytechnique de Toulouse

Université de Toulouse

Habilitation à Diriger les Recherches

in Computer Science

Defended by

Géraldine Morin

3D Models for...

on January, 14th, 2014

Jury :

President :	Stefanie Hahmann	Professor	INPG - INRIA
Reviewers :	Pere Brunet	Professor	Polytechnical University of Catalonia
	Klara Nahrstedt	Professor	University of Illinois
	Marc Neveu	Professor	University of Burgundy
Examinators :	Christophe Godin	Research Director	CIRAD - INRIA
	Vincent Charvillat	Professor	University of Toulouse

Acknowledgments

First I want to thank the members of the jury for accepting to participate in this committee: in particular, Klara Nahrstedt, Professor at the University if Illinois, Pere Brunet, Professor at the Polytechnical University of Barcelona, and Marc Neveu, Professor at the University of Burgundy for reviewing this manuscript, Stefanie Hahmann, Professor at the University of Grenoble for accepting to be the President of the committee, and Christophe Godin, Research Director in Cirad-INRIA, and Vincent Charvillat, Professor at the University of Toulouse.

Then, I want to thank all my research partners, students and colleagues who I have been working with; Axel Carlier, Wei Chen, Frédéric Courteille, Christophe Dehais, Jérôme Guénard, Sébastien Mondet, Phuong Nghiem, Dang Quoc Viet, Shanghong Zhao, Minhui Zhu whose work have been the building blocks of this document and my colleagues Frédéric Boudon, Vincent Charvillat, Jean-Denis Durou, Romulus Grigoras, Pierre Gurdjos, Sandrine Mouysset, Wei Tsang Ooi for the shared collaborations presented in this documents. I also want to thank the students whose work does not appear here for coherence reasons, Xavier Delaunay, Pascaline Parisot, Viorica Patraucean, Clovis Tauber. A special thank to Sylvie Chambon for her encouragements and help –Sylvie seems to be the perfect name to highly contribute to running the research lab! Thank you to our magic secretaries as well.

I also want to thank the persons with whom I share the work place with on a daily basis that make going to work such a nice experience. Also, I want to thank the members of the IMA department for welcoming me so kindly in Toulouse a little more than 10 years ago, who still stand and support me, even if I am grumpy.

I deeply thank Hans Hagen that encouraged me not to give up, and Ron Goldman who gave me the opportunity to get a new and enthusiastic start. All along, I need to thank many colleagues have been really supportive, and providing a great and motivating environment. A special thank also goes to the french geometric modeling community *GTMG (Groupe de Travail en Modélisation Géométrique)* for being such a motivating group to belong to.

Of course, I also need to thank the ones that provide some sense, motivation, support and fun besides work: Henrik, and our domestic team, my family and my friends. Thank you.

Contents

Keywords: Geometric Modeling, Multimedia, Shape analysis, Tracking, 3D Compression, 3D Transmission, Plant modeling.

Abstract:
The use of 3D models as multimedia content is spreading. However, even if online 3D models have been introduced, for example in e-commerce applications, 3D content is still marginal compared to images and videos. Virtual 3D objects are not yet very popular in common place applications and their use is confined to dedicated environment, like CAD-CAM applications. Indeed, 3D models need developments and tools to be handled more easily. This dissertation presents different tools to ease the use of 3D models as multimedia content: for creating, manipulating, and sharing virtual models.

In a first part, we derive image based techniques for creating 3D models. Prior knowledge on the model is assumed to reconstruct a realistic 3D virtual model from a single image. For the manipulation of 3D content, a tracking algorithm of a 3D model is proposed. Then, an analysis tool detects similarities within a 3D parametric model. Interactions with a single virtual object are also studied and techniques to simplify user interactions are developed. Finally, we address the problem of transmitting these models. We propose a compact and progressive model for plants and develop streaming strategies specific for such 3D progressive content over lossy networks. These streaming strategies are further used for previewing remote 3D objects, and a framework for navigating in 3D virtual environment with a light device is proposed.

Chapter 1

Introduction

1.1 Global context: the use of 3D models today

The field of geometric modeling started to develop around 1960 in the automotive industry, as computers started to offer the prospect of having a single, numerical model, from the conception of a shape to its machining. In [Farin 2002b], Farin gives a very detailed history of geometric modeling, starting at the Roman era, and explains that polynomials where chosen first by de Casteljau, and later and independently also by Bézier, as a numerical model for representing freeform curves and surfaces. Polynomial functions were expressed in the Bernstein basis, defining a Bézier curve (a polynomial), or surface by its control points (the coefficients in the Bernstein basis). Next, piecewise polynomials where considered to model more complex shapes (as opposed to increasing the degree) and also to keep modifications local to the edited control point. Their canonical representation as B-splines was already used by Schoenberg [Schoenberg 1969] in the context of smoothing statistical data. The GD (Geometric Design, also known as geometric modeling) community developed and provided different curve and surface models and their associated tools for generating free form objects. In 2001, the SIAM/GD conference[1], the business meeting (where all conference attendees are invited to participate) was about opening the thematic of the Geometric Design group. The idea was that a lot of the initial scientific advancements developed in the CAGD (Computer Aided Geometric Design) community had made their way into industry: NURBS (Non Uniform Rational B-Splines) was accepted as the standard for parametric models and subdivision surfaces were already making strides for content creation in the entertainment industry (the 1997 Pixar's short film Geri's game [deRose 2000] was the successful test-bed for a modeler based on subdivision surfaces, see Figure 1.1). The question was clearly whether geometric design was still relevant and sustainable as a dynamic and thematically large enough research field. Well, I believe the wheel has turned: lately, 3D modeling is facing new challenges, namely becoming part of our daily life, and being to building block for popular, common place applications.
The evolution of the use GD is tightly bound to the development of new tools and technologies. We briefly review the history of 3D modelers, displays, graphic cards, sensors and 3D printers and their impact on the evolution of models. The context where these models are used is also a determining factor: in CAD/CAM applications, representations are required to be precise, and in the entertainment applications, representations may be less exact (or within a tolerance) but the quality of the rendering and the speed of interactions are important.

The use of **3D modelers** has been moving from professional contexts to common place applications. Classical CAD/CAM software providers have developed new products. Dassault, for example, provided *Imagine And Shape* that uses subdivision surfaces for prototyping, in particular for avoiding issues when changing the topology. The goal remains to have an intuitive

[1] SIAM, the Society for Industrial and Applied Mathematics, is organized in thematic groups, among them the Geometric Design (GD) group.

Figure 1.1: *Geri's game*, a short film made by Pixar in 1997, was a successful try at using subdivision surfaces as the underlying model [deRose 2000]. Above its scientific impact, Geri's game won many awards `http://en.Wikipedia.org/wiki/Geri%27s_Game#Awards`.

interface, hiding the underlying mathematical tools to the designer. However, designer using classical CAD/CAM software are professional users, or have received training to use a given modeling software efficiently (Figure 1.2, left). In medicine, visualization techniques evolve rapidly and now use 3D models and visualization tools, as for example in Geneva (see Figure 1.2, right). But the use of 3D models is now going beyond the scope of these classical professional users. As the technology evolved drastically, the 3D content creation became accessible to hobbyists. In terms of 3D design, *Second life* appeared in 2003, allowing users to create and share 3D content. End of 2009, the Khronos group proposed the royalty-free WebGL standard for including low-level 3D graphics into web browsers[2]. WebGL should drive the development of innovative applications for the consumer (e.g. e-commerce), hobbyist (see e.g. `tinkercad.com`) and professional markets (e.g. visualization of contents on mobile devices). Ordinary people may contribute 3D content in *Google Maps* by designing a 3D model of his/her favorite buildings[3].

The same evolution, from very specialized users via industrial consumer, to personal use is happening for **displays**. A first implementation of a CAVE was proposed at the beginning of the 90s [Cruz-Neira 1992, Cruz-Neira 1993], and research institutions started to be equipped with such immersive virtual devices (e.g. Figure 1.3, left). The market for such devices has moved from high end mechanical design workshops and niche markets to everyday use. You can now see 3D movies in regular movie theaters, and even buy your own equipment to play 3D games at home. For example, *NVIDIA* has launched its second generation of 3D glasses and displays (Figure 1.3). The Nintendo 3DS, a hand-held entertainment device for kids (over 8 years old!), was launched in 2011 providing 3D content through a stereoscopic display without the need of any glasses.

[2] `http://www.khronos.org/webgl/`

[3] see *3D Warehouse* `sketchup.google.com/3dwarehouse/` from *Google*

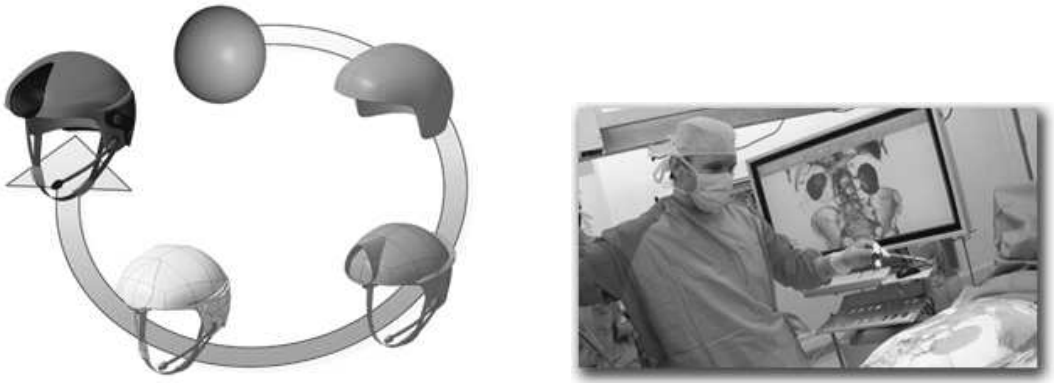


Figure 1.2: Professional use of 3D software: on the left, an example of modeling a helmet with Imagine and Shape (source `http://fa.wikipedia.org/wiki`) ; on the right, Pr. Morel starting an operation: a 3D model of the internal organs have been reconstructed and studied before the surgical operation (source: HUG (Hopitaux Universitaires de Genève), in french `http://www.visceral-surgery.ch/visceral/index.php?option=com_content&view=article&id=283&Itemid=160`

Figure 1.3: From academic use of 3D immersive devices, to home 3D displays: on the left, a modern immersive CAVE, *aixCAVE*, from Aachen University (source `http://www.rz.rwth-aachen.de/aw/cms/rz/Themen/Virtuelle_Realitaet/infrastructure/~tos/`); on the right, illustration of a home 3D screen and glasses, like NVIDIA 3D vision system (`http://www.nvidia.com/object/3d-vision-main.html`) (photo from `http://www.digitaltrends.ro`)

The improvement of the display has been also possible thanks to the improvement of the realism of the rendered images. In the early 80s the first hardware implementation of the geometry pipeline, originally running on software, was the base for the SGI (Silicon Graphics Inc.) rendering engine. At first, SGI produced 3D graphics display terminals but moved rapidly to producing specialized workstations for Graphics. In the 90s, the hardware support required to perform the manipulation and rendering of quite complex scenes has become accessible to the consumer market. Advances in the efficiency and versatility of the **Graphic Processing Units** (GPU) accompanied the development of these new display technologies. The game industry has been a driving force in the evolution of graphic cards. The late 90s saw the beginning of a mini-revolution in computing, with the release of the first discrete graphics chips for personal computers (3dfx Voodo in 1996, Nvidia TNT 1998). The massive popularity of

their successor accompanied the shift of the video game industry toward real-time 3D rendered titles. The creation of assets (which nowadays accounts for most of the cost of these products) spurred the demand for improved, more intuitive and efficient geometric design modelers. This eventually pushed the hardware designers to produce even more capable chips, driving a virtuous loop that continues nowadays. Graphic cards not only became more efficient, they also became programmable, opening the door to new rendering pipelines and therefore to models in concurrence with meshes, the only model natively supported. Point based models that had been already proposed in 1985 [Levoy 1985] but got a renewed, significant attention in the research community around the years 2000 as point-based model could then be rendered real time on the GPU. Other authors have proposed new models to benefit from the new features of graphic cards, like Loop et al. who create a smooth surface by computing parametric patches on the tessellation shader [Loop 2008, Loop 2009]. GPU performances also leads to renewed recent interest in the computer graphics community for real-time / interactive global illumination simulation have made ray tracing approaches popular again: those techniques work with representations different than meshes, like implicit models, and require space partitioning structures (e.g. kd-tree or bounded volume hierarchies)[4]. But displays and GPUs are not the only 3D technologies to have gone under recent major changes.

Sensors apprehend or digitize the surroundings around us. Of course, as for displays, the innovation started in specialized environment. For example, the *3D Dome* was developed in Carnegie Melon University in 1995 for creating what they call virtualized reality sequences, that is, $3D + t$ data digitizing a real scene (Figure 1.4, left). The *Digital MichaelAngelo project* happened during the 1998-99 scholar year (Figure 1.4, right); digitizing museum pieces has since spread. New 3D applications are developing very fast. Everyone can for example create its own 3D avatar from, portraying your own head (see Figure 1.5, left). The movie industry has been a big consumer for new sensor technologies. The production company Pixar laid a small renaissance of the animated feature film (with nearly 15 movies since Toy Story in 1995, the first computer generated feature film). Nowadays its renderer PR Renderman[5] is the leading commercial renderer for Hollywood productions, and CGI animated movies (with the competing studio Dreamworks) are a mainstay of the cinematographic landscape. Prominent Hollywood productions also consolidated mixing virtual and real imagery as an integral part of modern enternainment artpieces (the special effect company ILM[6] contributed to Jurassik Park, the Star Wars saga, Men in Black series, Weta Digital contributed to the Lord of the Ring saga, Avatar). These productions require new ways of creating and managing 3D assets. To note, this trend was also made possible by advances in Computer Vision, which allowed such things as tracking the location of the camera in real footage (making a realistic integration with rendered image easier), acquiring the lighting environment of the scene and more recently tracking the articulated motions of real actors to replace them by virtual avatars. Besides, new types of cameras are offering to create 3D content: the first time-of-flight camera for civil applications *Z-cam* came out in 2000, as the semiconductor processes became fast enough for such devices `http://en.wikipedia.org/wiki/Time-of-flight_camera#cite_note-ZCam_history-2`. Today, the price of time of flight camera's make them usable for private use. In 2009, Nintendo launched the Wii console, able to detect 3D motion and in 2010 Microsoft launched the *Kinect*. The Kinect, which use and price is meant for regular private consumers, has also been used for different applications.

[4] For a example, see `http://madebyevan.com/webgl-path-tracing/`

[5] `http://renderman.pixar.com`

[6] `http://www.ilm.com`

For example, Oikonomidis et al. [Oikonomidis 2011] use it for hand motion tracking, and Changa et al. and Chopping et al.[Changa 2011, Choppin 2012] propose to benefit from a Kinect sensor to guide disabled young adults (Figure 1.5, right). The evolution of sensors, and the fact that they are affordable, broadens the average consumer capability of capturing the surrounding real 3D world. However, we need to point out that our interaction with screens has also changed a lot in the last 10 years. Tactile screens offer new ways of interact. Although touch-screens existed for a long time: a short article appear as early as 1965 [Johnson 1965]), the first touch screen phone, the *IBM simon*, appeared in 1993. Nowadays, touch screens are everywhere; small kids' ability to play on tactile devices is a witness to how intuitive touch based interactions are.

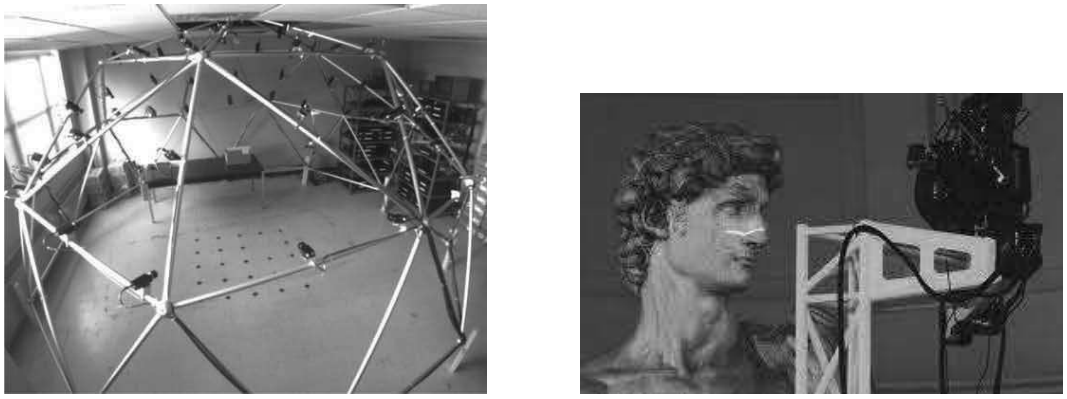


Figure 1.4: Acquisition of *3D* or *3D+t* content in specialized environments: on the left, the Dome from CMU (source: `http://www.cs.cmu.edu/~virtualized-reality/page_History.html`); on the right, scanning of Michelangelo David's head (source: `graphics.stanford.edu/projects/mich/`)

To wrap up our horizon tour on new devices, we have to mention another important tool (Figure 1.6). The first **3D printer** has been developed by C. Chuck Hull [7]. Early efforts at University of Bath by Adrian Bowyer and later developments made by the RepRap community[8] to build a self-replicating 3d printer using patent-free technologies recently raised the industry and public awareness toward 3D printing technologies. 3D printing (which builds a solid object by layering microscopic slices of material, such as plastic, concrete or metal) has now been successfully used as an industrial design tool for rapid prototyping, in stop-motion animation for figurine making (e.g. in the movie *ParaNorman*[9]), in NASA for rocket engine parts and also for food to be printed in space [10]. The joint emergence of the maker movement, the hobbyist market and crowd-funding platforms lead to a rapid explosion of 3D printing products since the middle of the 2000. Many companies are now offering 3D printers targeted to the consumer market at increasingly affordable prices. Since the beginning of the century, the price of 3D printers have dropped significantly and they are common place. This technology widen the range of use of 3D modeling solutions, 3D assets sharing and manipulation. 3D printers have been labeled disruptive technologies[11], that is, supposed to significantly make a difference.

[7] `http://www.pcmag.com/slideshow_viewer/0,3253,l=293816&a=289174&po=1,00.asp`
[8] `reprap.org`
[9] `http://www.wired.com/design/2012/07/paranorman-3d-printing/`
[10] `http://www.space.com/22568-3d-printed-rocket-engine-test-video.html`
[11] An article of the Economist `http://www.economist.com/node/18114327` compares 3D printers to steam

Figure 1.5: New common place application based on *3D* sensors: on the left, the software EasyTwin© from digiteyezer allows to acquire and print (source: `http://www.digiteyezer.com`); on the right, using the Microsoft's Kinect as Navigation Aids for Visually Impaired [Choppin 2012] (photo from `http://www.ubergizmo.com/2011/03/visually-impared-kinect/`)

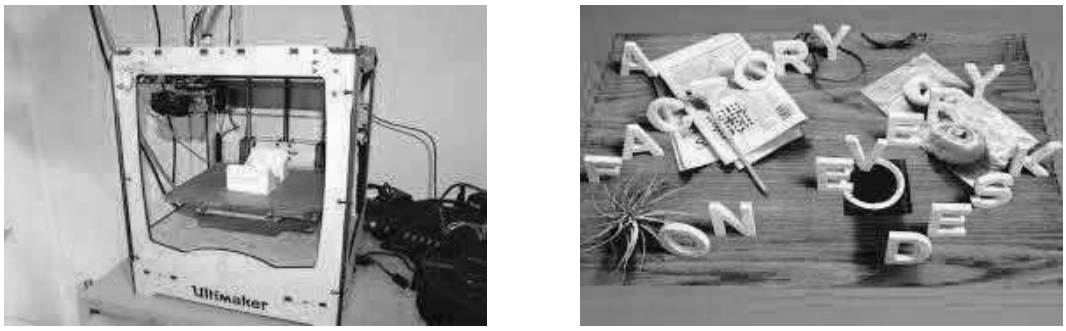


Figure 1.6: 3D printing: The ultimaker 3D printer (left), and some printed (right). Figure 1.5 (left) also shows some printed figurines.

To sum up, 3D has become a really hot topic, as claimed in the call for paper in the Hot 3D conference[12]. The 3D community is obviously facing new challenges: providing new tools for creating 3D models, manipulating them (comparing, editing) and being able to share 3D content.

machines.

[12] `http://www.hot3d.org`

1.2 Different representations for 3D models

We have seen in the previous paragraph the general context, and some new, common place technologies and applications for which 3D models are required. We now review the different classical representations available for modeling 3D objects and their advantages and drawbacks. To model 3D objects it may seem natural to use **volumetric data**. Volumetric modeling includes particle representations [Sims 1990] which is particularly adapted for fuzzy or fluid objects (usually in motion) like smoke or water. Other solid volumetric model are 3D meshes that are adapted for volumetric finite element computation or 3D visualization [Pain 2001, Gumhold 1999, Freitag 2002]. However, since the appearance of 3D object mainly depends on its exterior, a representation of the surface of the object, called boundary representation, is used in most applications. In our work, we only use surfaces for modeling 3D objects.

As mentioned in the previous paragraph, **parametric surface models** are the most commonly used in CAD/CAM software [Farin 2002a]. NURBS [Piegl 1997] has become a standard for the parametric design software, and is now used by most software today, like *CATIA* from Dassault System[13] or *Creo* from PTC (Parametric Corporation Technologies)[14]. These parametric surface models are either piecewise polynomial, or piecewise rational tensor product surfaces. They are also called free-form surfaces and are intuitively controlled by a set of control points and are generally smooth surfaces. Their smoothness is controlled by the knot vector and the degree of the underlying polynomials even though in practice degree 3 or 5 polynomials are used in most cases. Formally, a NURBS surface patch is a piecewise rational map, or more exactly the mapping of a 2D domain into 3D. The parametric representation offers the advantage to be able to easily generate points on the surface by simply considering the image of points in the domain. Moreover, information about the derivatives of arbitrary order is easy to compute (up to the continuity for a parameter value corresponding to a knot). The correspondence between the mathematical properties of the function/mapping and the surface depends on the parameterization (e.g. [Goldman 2002]). One way to keep a good correspondence, is to ensure that the parameterization is as regular as possible; for that, if the knot vector is regular, the control point should be regularly spaced [Farin 2002a, Tauber 2004] or the parametrization may be adapted considering arc length, or centripetal parametrization [Foley 1987]. Alternatively, geometric continuity may be used [Barsky 1984, Peters 2001]. In general, parametric models are used in applications where the quality and accuracy of the surface is important, both in terms of measure on the model and but also in terms of smoothness.

Mesh models consist of a discrete set of vertices (the geometry), and a topology (a graph) defining the faces and edges of the surface. The minimum property required for a mesh properly model a surface is to be manifold (that is at every interior point locally homeomorphic to a disk) . Meshes are from far the most commonly used models, since they can be derived from a set of sample of points on the surface. Any arbitrary mesh can be easily triangulated (each face has three vertices). Triangulations are the only models to be natively rendered by graphic cards. Scanline rasterization algorithms for triangular facets are well understood and were implemented by the early dedicated hardware components, which further cemented the triangle mesh as the canonical surface representation. Unlike parametric surfaces, meshes are only continuous surfaces, with discontinuities of order 1 on the edges. Meshes are therefore used in

[13] http://www.3ds.com/fr/products-services/catia/
[14] http://www.ptc.com/product/creo/

applications where the rendering speed in the most relevant factor, and accuracy and smoothness is less important. However, a lot of papers have proposed techniques to evaluate approximate derivative information on a mesh, e.g. [Rusinkiewicz 2004, Morvan 2008]. Moreover, from the visualization point of view, rendering algorithms like Gouraud (or Phong) shading consider a set of continuous derivative of the mesh surface which smooths the appearance of the mesh. Modern rendering techniques such as normal mapping further decouple the 1st order derivative information (which can be made very detailed at a relatively low cost) from the geometrical information, making even low density meshes visually pleasant. One can also consider fine enough meshes so that the projection of a triangle on a display does not exceed a small number of pixels. **Subdivision surfaces** (e.g. [Warren 2001]) offer refinement operators on a mesh, so that a mesh can be refined to an arbitrary resolution and converges to a smooth surface. When the mesh is regular (triangulation where each vertex has valence six, or a quad mesh where each vertex has valence four) and for classical subdivision schemes (like Loop [Loop 1987] or Catmull-Clark [Catmull 1978]) or well-chosen schemes [Sabin 2010, Barthe 2011], the subdivision process actually converges to a parametric surface, namely a Box-spline. In the case of irregular vertices, particular rules are required, and an analysis of the behavior of the subdivision is done to study the smoothness of the resulting surface [Dyn 2002, Peters 2008] . Subdivision surfaces offer a good compromise between mesh and parametric representations. They naturally provide a multiresolution setting, and allow to easily work with surfaces of arbitrary topology. They have been used in the movie industry for more than 15 years now, and have also make their ways into CAD software. Recently, Pixar has released its modeling package into open source [15].

Another discrete representation for 3D models is the **point-based representation**. Points based models are a set of vertices on the surface. As mentioned in the previous section, point-based models have been around since 1985 [Levoy 1985] but got a growing attention since the years 2000 (e.g. [Pauly 2002, Gross 2011] as they could then benefit from programmable GPU to be rendered in real time [Guennebaud 2003]. Point based model can be presented intuitively in two ways. First, we may say that if a mesh is refined enough so that the projection of its faces on the screen just covers a couple pixels, then the topology information may be spared, and only the vertices can be kept. Neighboring vertices of a given vertex may be retrieved if necessary, simply by checking nearby vertices (for more details see [Pauly 2003a]). A second way is to consider point set surfaces as the generalization of meshes where one has reduced the continuity of the surface. Instead of having a C^0 continuity surface, we now have a C^{-1} surface. In point set surfaces, the vertices are usually attached a radius, and a normal. Point based models are the raw data output by range 3D scanners, which make them ideal for early visualization applications, or for merging many scans into a single representation. Moreover, the lack of topology may also be an advantage for representing object with particularly complicated topology, like a tree [Guennebaud 2004], or for keeping independent geometric data, for example for streaming [Mondet 2007]. Most of the time, point based models are stored in a space partitioning data structure (like octree, or K-d trees), to ease the access to neighbors.

Another classical representation is based on **implicit surfaces** [Bloomenthal 1997]. Implicit surfaces are defined as the iso-surface of a potential function defined in $\mathbb{R}^3$. These surfaces are particularly interesting since they combine a volume and a surface representation. However,

[15] `http://graphics.pixar.com/opensubdiv/`

their main drawback is that sampling points on the surface is a difficult task. A triangulation may be recovered after dicing the volume into a voxel grid and running a so-called marching cube algorithm. Thus, a classical rendering through a regular graphic pipeline is neither easy, nor efficient. As an alternative, rendering based on ray tracing is adapted for implicit representations (but not real time –yet).

Finally, **imaged based representations** (IBR, for Image Based Rendering) have been proposed mainly by the rendering community. Many representations and techniques were invented for image-based rendering (IBR), for example, the Lumigraph [Gortler 1996] and the light field [Levoy 1996], mosaics and panoramas (e.g. [Shum 2002, Peleg 1997, McMillan 1995]), and texture maps (e.g. [Debevec 1998, Buehler 2001]). A detailed survey is conducted by Zhang [Zhang 2004]. They use a set of images, possibly associated with depth maps or disparity to represent geometry and texture. Some approaches have considered multiresolution or multi-layered approaches [Shade 1998, Woodford 2005]. When the represented geometry corresponds to a single object, its image representation is called *impostor*. A detailed survey on this subject is presented by Jeschke and Wimmer [Jeschke 2005]. Schaufler [Schaufler 1998] also proposed layered impostors. In some work, impostors can also be textured depth mesh (TDM), as for [Aliaga 1999].

We have reviewed the classical 3D representations, and pointed out their advantages or drawbacks and applications for which they are preferred. In the work presented here, different 3D models have been used, in order to benefit from their properties. Parametric models and parametric surfaces are used in two different settings: Chapters 2 and 3. In Chapter 2 about creating 3D models, parametric surfaces are used for *shape from shading* [Courteille 2006a, Courteille 2006b]. The choice of the underlying parametric reconstructed model not only insures the reconstruction of a smooth surface, but it also reduces the number of unknowns, which is necessary for the computability of the solution. In Chapter 3, the model is the starting point, as we are interested in identifying parts of a classical CAD model that are similar up to an isometry [Dang 2012, Dang 2013]. Moreover, the parametric representation eases the computation of characteristics of a sample point on the surface, which is an advantage. We have used simple mesh models with textures in Chapter 3 when extracting semantic information from user interactions [Nghiem 2012, Nghiem 2013]. We choose these models for their popularity, since they are the most likely to appear on e-commerce applications. Meshes are also the most popular representation for 3D objects in virtual environments, so, in Chapter 4, we have considered meshes for the transmission of 3D models [Cheng 2007, Cheng 2011, Zhao 2013]. We use for that the classical progressive mesh representation from Hoppe [Hoppe 1996]. In this document, point set surfaces are used in Chapter 3 in a tracking algorithm [Dehais 2006, Dehais 2010]. As we assume that a 3D model of the object is known, it is actually not necessary to define the topology of the set of point of the object. Having just a set of point saves the definition of the topology, and allows to directly consider the scan of an object as an input to our algorithm. We actually also have used point-based model in the context of 3D streaming (Chapter 4), but this work is not presented here for keeping the material reasonably concise (more details can be found in [Mondet 2007]). We also use in this document, both in Chapter 2 and Chapter 3, a **curve based representation for representing plants**, and in particular plant branching structures. In Chapter 2, we create a realistic model of a plant from biological knowledge of its species and a single image [Guénard 2009, Guénard 2010, Guénard 2011, Guénard 2012, Guénard 2013b]. In Chapter 4 we propose a compact and progressive representation for trees branching sys-

tems and use it for efficiently transmitting plants models over a potentially lossy network [Mondet 2007, Mondet 2008, Mondet 2009b, Doran 2009]. In section 4.6 , we proposed to use image based representation to ease access to NVE (Networked Virtual Environments) for light clients. We consider IBR first to limit the size of the data to be sent, and also to simplify the rendering task for the light client [Zhu 2011].
Note, that we do not consider stereo vision, although these representation are already quite advanced in the context of democratizing 3D, e.g. 3D displays are mostly based on stereoscopic images. Our work focuses on actual 3D models that do not depend on the viewpoint. However, in our context the interactions between the three dimensional world and their images (or sequence of images) in 2D are important, as we highlight in the next section.

1.3 3D and 2D

Considering the rendered image of a given 3D model is necessary since we apprehend the 3D constructed models mostly through flat 2D screens. However, handling a 3D model is very different than working with 2D content. A 3D object appears on the screen as a 2D image, but their 3D nature is still present. First, viewing parameters may be chosen and changed: the viewpoint, of course, is free and allows navigation around the 3D model. Also, lighting parameters, and even the color of the model may be changed. So, interacting with a 3D model is much richer than just viewing a static image of this model. In Chapter 3, we make a first step towards simplifying user interactions with 3D content, by linking viewing parameters to simpler information: a text description. In Chapter 4, we propose an alternative to previewing a 3D model in a 2D video, by rather displaying well chosen views of the 3D content while downloading the model. In both cases, a simpler representation (text or image sequence) is used to restrict the access to 3D, in the first setting to limit the degrees of freedom in navigation, in the second to leave some time for downloading the model while adapting the transmission to the ongoing interaction (simulation of a video).
Moreover, images and videos have already made their ways in our daily life (we use in the following the term *2D content* to denote both images and videos). Since numerical cameras, and video cameras are now included in every mobile phone, the diffusion of 2D data has been very demanding on solutions for storing analyzing and sharing this content (100 hours of video are uploaded to YouTube every minute[16]). These 2D content, images or videos representing the real 3D world, offer almost unlimited resources for apprehending 3D models. We indeed start from images for creating 3D content in two different applications of Chapter 2. Moreover, in Chapter 3, the proposed tracking algorithm finds the position of a 3D model within a video. Finally, in Chapter 4, we consider *peer-assisted-rendering*: in a popular NVEs where numerous clients evolve, users with limited resources request pre-rendered 3D objects to users with spare rendering cycles in order to simplify their own rendering.

These interactions between 2D and 3D content have led to convergence between different research communities. The Computer Vision and Computer Graphics communities are getting closer; topics like stereovision, and 3D reconstruction are really shared between these communities. Moreover, communities like the computational geometry, signal processing, and Multimedia community are also getting involved in 3D modeling. For the signal processing community, they have been applying their methods to 3D data: for example, much work on

[16] `http://www.youtube.com/yt/press/statistics.html`

compressing 3D data has been done by signal processing researchers (see Chapter 4, section 4.3.3). Similarly, 3D content is a new media to store, edit and distribute for the multimedia community. As some multimedia technique may apply the same way to all media, some specific approach may need to be deploy for 3D content (see Chapter 4, section 4.2.1).

A final note on 2D content: this document will not mentioned some work done on 2D images (compression [Delaunay 2007a, Delaunay 2007b, Delaunay 2008a, Delaunay 2008b, Delaunay 2010] and *2D+t* image sequences (tracking [Tauber 2004, Parisot 2004, Parisot 2005a, Parisot 2005b]) for coherence reasons, as this document focuses on 3D content.

Organization of the manuscript

To conclude on the concerns raised in 2001 in the SIAM/GD meeting: we have seen that a brand new challenge for the geometric modeling community has emerged: democratizing the use of 3D content. As discussed, some collaborations with different research communities will certainly be relevant. 3D content will still be used by CAD companies, and the entertainment industry, in both context requiring new models, tools, methods and developing new dedicated material. However, 3D content will also be more present as multimedia content in our daily life. To that end, we need to develop new tools for creating, manipulating and sharing 3D content. In Chapter 2 will present two dedicated methods for creating 3D models from a single image. For the use of 3D models, new, intuitive and efficient tools will be necessary. Chapter 3 present three contributions for manipulating 3D content. The first one proposes an original tracking algorithm to follow a 3D object whose model is known through a video sequence. The second one analyzes 3D models for identifying partial similarities in parametric models. The third contribution simplifies the interactions of regular users with an online 3D model, for applications e.g. in the context of e-commerce. Finally, Chapter 4 addresses the problem and the specificity of the transmission of 3D content. We consider, and develop, progressive models for plants, and derive streaming strategies considering possible packet losses.

Chapter 2

Creation of 3D models

Contents

2.1 Introduction

The first step for an extended use of 3D models is to generate 3D content. There are two ways to model 3D objects: Computer Aided Design (CAD) or digitizing an existing model. The first way is to generate virtual objects. This part is concerned with synthesis. These virtual object may be intended to be produced, for example, when creating a new car model using a CAD system; but, these object may also be intended for staying virtual, like our favorite figures in animation movies, monsters in video games, or explanations added to a tool in the setting of augmented reality. In both setting, the 3D object is designed using a software, depending on the considered application; some examples of modeling software are Blender (an open Source software), Maya or 3ds max (from Autodesk, the first one rather for the film industry, to second one for the game industry), Rhino (from *Robert McNeel & Associates*), or CATIA (from *Dassault System*). For using a CAD software however, a particular training is necessary. Moreover, the process of creating a 3D model from scratch is quite time consuming, and requires not only good technical skills from the designer, but also a good domain knowledge, or artistic skills.

The second way is to model real existing objects. For that, 3D sensors may be used. We already have talked about digitizing museum pieces with a scanning device e.g. [Levoy 2000]. In medicine, 3D models may also be reconstructed from 2D slices or volumetric sensors, like VCT (Volumetric Computed Tomography). These methods are classified as **active** reconstruction methods, since they interact with the object. On the opposite, image based reconstruction methods are also called **passive** methods.

In this Chapter, we present the results of two passive methods for generating 3D objects. In both cases, we assume some knowledge on the final object, in particular by restricting the 3D model. First, we briefly review general reconstruction from multiple images.

The work presented in this chapter has been developed in the context of the doctoral work of Jérôme Guénard (plant modeling), and Frédéric Courteille (Shape from Shading). For the first part (section 2.2), more details can be found in Jérôme Guénard's dissertation [Guénard 2013a], co-advised with Pierre Gurdjos and Vincent Charvillat. The plant modeling work was done in collaboration with Frédéric Boudon from CIRAD. The contributions have been published in [Guénard 2010, Guénard 2011, Guénard 2012, Guénard 2013b]. The second part was developed in the context of Frédéric Courteille's Ph.D.; I collaborated with him and his advisor Jean-Denis Durou for a common publication [Courteille 2006a].

3D reconstruction from images

Intrinsically, 3D reconstruction from a set of images is a difficult problem. A realistic 3D reconstruction of a small 3D object can be generated from a set of well chosen images, taken in good conditions, with careful calibration. Seitz et al. [Seitz 2006] propose benchmarks and comparison of different reconstruction algorithm are given on the corresponding website `http://vision.middlebury.edu/mview/eval/` (Figure 2.1). The minimal set of images considered contains 16 images. Images with shadows have been removed from the data set and radial distortion removed. The parameters of the camera are given for each image. This benchmark

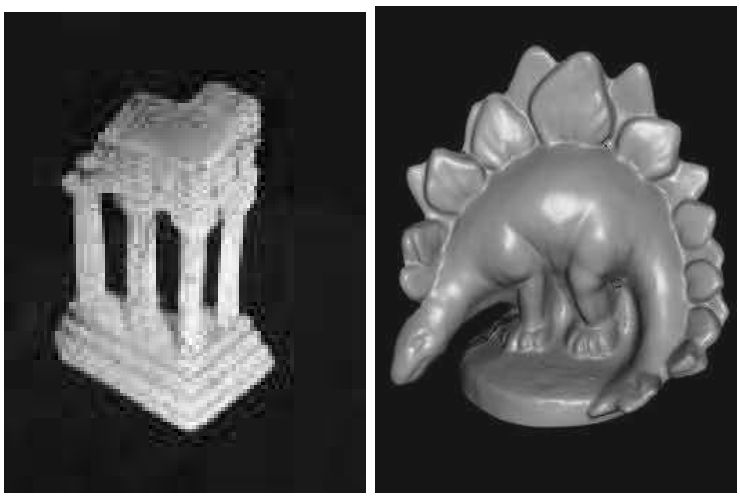


Figure 2.1: The two 3D models proposed as benchmarks for 3D reconstruction algorithm [Seitz 2006].

shows that 3D reconstruction can be achieved in very controlled environment.
Other work achieved a good 3D reconstruction from images taken in arbitrary conditions: Goesele et al. [Goesele 2007] use online photos of famous monuments to reconstruct a 3D model. The number of input images is very large (several hundred images). This work applies to famous buildings, for which many images are available.
A different approach reduces the degrees of freedom in the reconstruction by introducing domain knowledge on the model. For example, Bey et al. [Bey 2012] recover some 3D objects in an industrial environment (from 3D point clouds) assuming the 3D object belong to a set of

predefined shapes (cylinders and spheres) since they know they are in an industrial setting where the 3D object are mostly pipes.
In this chapter, we use a similar approach. First, in the context of plant modeling, we use L-systems, a classical generating method. However, our reconstruction is based on the shape given by an image of the plant. Using the image insures the realism of the model, avoiding the fractal-like natural regularity of plants generated by L-systems. Moreover, the particularity of plants is to have a very complex topology, and also to be highly self similar. Correspondences are hard to find, and there is not much depth coherence. For these reasons, we consider a single image. Then, we show results on reconstruction from a single image in a different context.
In the following, we first review the classical models for plants and previous work on modeling plants from images.

2.2 Modeling plants from one image

2.2.1 Previous work

2.2.1.1 Plant modeling

Plants are important and common objects of our world. Just as in the real world, plants help create pleasant and realistic virtual environments, especially those involving natural scene. Realistic modeling of plants are crucial applications such as virtual outdoor scenes like virtual botanical gardens, where users are expected to inspect a plant closely and possibly interact with plants. Previous work has focused on how to accurately model a plant, e.g. [Remolar 2002, Bloomenthal 1985, Prusinkiewicz 1990, Prusinkiewicz 2001]. Realistic and detailed plant mesh models can require up to hundreds of thousands of polygons. Remolar et al. [Remolar 2002] estimated that a plant generated by XFrog[1], a well known plant modeling platform, can consists of 50,000 polygons to represent the branches. The plants can have 20,000 or more leaves, which themselves consists of polygons. Neubert et al. [Neubert 2007] reported the plant models that they used consists of up to 555,000 polygons. These numbers are for a single plant.
As plants are, by nature, very self similar, associating redundant parts within a plant model allows to simplify the model, and to a more compact numerical representation. For example, the leaves of a given tree maybe represented by a small number of models, instantiated several times. The pioneering work of Lindenmayer [Lindenmayer 1968] proposed to formalize the redundancy within plants by modeling plants using *L-systems*. The idea is to model a plant as a formal language, thus following a limited, simple generating rules. This modeling technique has since became a standard for plant modeling [Prusinkiewicz 1990, Deussen 2005, Weber 1995]. However, a simple deterministic application of the generating rules of a L-system leads to very self similar plants, similar to fractals **??**. But plants are also characterized by their irregularities. Thus to create realistic plants, irregularities are necessary but need to be coded, leading to a compromise between the compactness and the realism of the model [Boudon 2006] (see Figure 2.3). Prusinkiewicz has proposed to use non deterministic rules in order to generate irregular plants [Prusinkiewicz 1990]. Some authors derive the probabilistic laws from botanical studies [de Reffye 1988, Chaubert-Pereira 2010] to generate a random, irregular instance of a plant. Other approaches consider physical constraints. For example, both Palubicki et al. and Runions et al. grow a plant within a given volume, where branches compete for filling the space [Palubicki 2009, Runions 2007].

[1] `Xfrog.com`

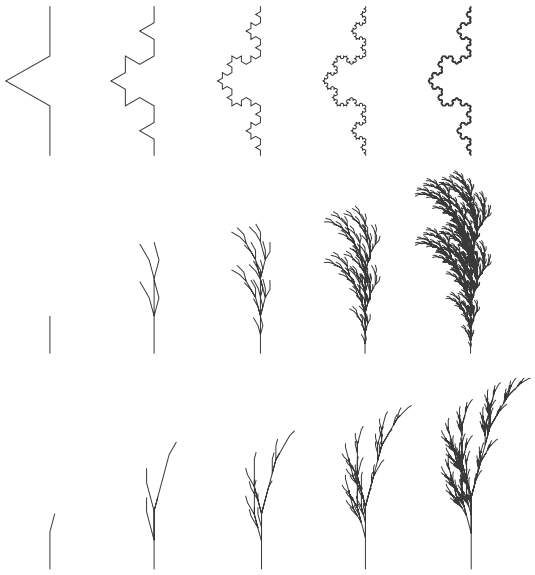


Figure 2.2: Generating plants like fractal, using simple deterministic L-systems.

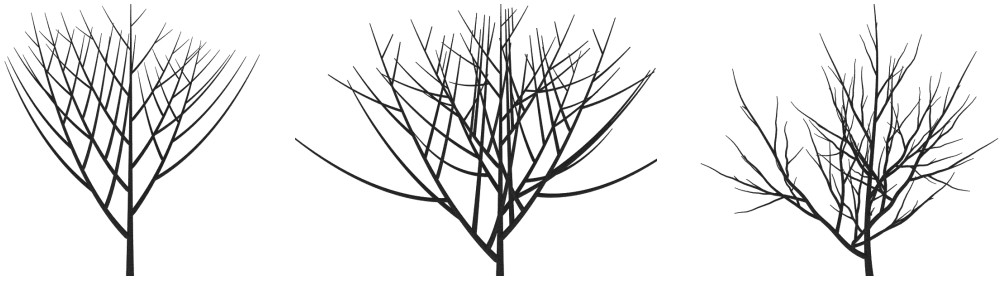


Figure 2.3: Three instances of plants modeled with the L-py software `http://openalea.gforge.inria.fr/wiki/doku.php?id=packages:vplants:lpy:main`. From left to rigth, as irregularities appears, the plant becomes more realistic.

These different approaches propose to generate models of plants using generative models corresponding to a species following some botanical or physical rules, and create a particular, irregular and realistic instance using external constraints. An alternative way to model a particular instance, is to model an existing plant. Plant modeling from one or several image proposes to model a particular instance of plants. In the next section, we review different approaches for modeling plants from images.

2.2.1.2 Modeling plants from images

Procedural method based on L-systems can model the growth of a plants, and follow its evolution. The methods based on images only give a model of a plant at a particular point in time, namely, when the image has been taken.

Some methods are based on 3D reconstruction of a set of points. For that, and because plants are very self similar, 2D point correspondences is challenging. Quan and al. first proposed to model simple plants from a set of images [Quan 2006]. The set of images need to be sufficiently dense for the 3D points to be extracted with a *structure from motion* algorithm. The plant consists in stem and leaves, so leaves are positioned using the reconstructed 3D points, and corresponding deduced. Tan et al. [Tan 2007] adapted the preceding method to tree modeling. Similarly, a set of 3D points is extracted from structure from motion. The 3D reconstruction is used two ways: first, the point set defines a volume, in which branches are generated. Second, visible branches are reconstructed. The structure of the branching system is extrapolated from the visible branches structure.

Other methods extract a volume from the silhouette of a tree in several images. In that setting, no 2D point correspondences is needed, and there may be fewer images. Both Reche-Martinez et al. [Reche-Martinez 2004] and Neubert et al. [Neubert 2007] extract a volume where they attach attributes. In the first paper, they generate real time volume representation of trees to be inserted in virtual environments. Neubert et al. generate a real 3D model of the tree. The branching system grows interpolating a set of particles withing the volume.

Finally, in a more recent work, Li et al. consider a video as the input, and generate a dynamic 3D model of the tree. They also derive dynamic properties of the 3D model to animate it through the wind, and also proposed to generate similar but different trees in order to create forest. Similar to Wang et al. [Wang 2006] analyze a set of images of trees of the same species and from this analysis to generate realistic model of trees of the same species.

In these approaches, a dense set or a sequence of images is required (point correspondences may be identified only of the set of images is dense enough). We now look at methods were a single image suffices.

2.2.1.3 Modeling plants from a single image

First, let us make a detour to approaches based on sketching; they are related to methods using a single image since they usually consider also only one viewpoint. Okabe et al. propose to sketch the shape of the branching system in 2D and use it to infer their position in 3D [Okabe 2005]. [Runions 2007], [Wither 2009] and [Talton 2011] ask the user to sketch in 2D the volume of the foliage. Runions et al. iteratively grow the branching systems to progressively fill the volume, whereas Wither et al. proposes an interactive multiresolution approach to progressively define the shape of the branches inside the volume. Talton et al. use a formal grammar (like L-systems) that they control to fill up the desired volume. Some other approaches mix image and sketching,

asking the user to draw on an image of the plant. For example, Liu et al. [Liu 2010] use the annotated image to reconstruct 3D points, and define a 3D embedding volume.
The following method are based on a single image, but as some require user interaction, they are indeed quite similar to approaches involving sketching. Tan at al. [Tan 2008] propose to reconstruct trees from a single image, but requires user interaction to outline the foliage volume and draw the visible branches. Using the visible branches shapes as pattern, a branching system is generated inside the volume 2.4. The method proposed by Zeng et al. [Zeng 2006] is using

Figure 2.4: The pipeline proposed by Tan et al. for reconstructing a tree from a single image. User interaction is required to segment the foliage and mark the visible branches (images from [Tan 2008]).

images of branching systems, and reconstruct a tree without the leaves first in 2D, and then embeds it in 3D space while defining a volume for the leaves.
The existing methods require some user interactions (method based on sketching, [Tan 2008]) or/and visible branches ([Tan 2008, Zeng 2006]). In the next section, we propose a fully automatic approach for reconstructing plants from a single image. We start with plants of a given species, here wines. We then generalize our methods to trees: for that, we need to get a plant in 3D and work with a multiresolution scheme. Visible branches are not necessary, as we assume a prior knowledge on the plant species, and use some botanical knowledge for creating the branch model. Thus, the generated model is biologically sound, and also realistic since it fits a existing instance (on the image).

2.2.2 Extracting a 2D skeleton

We have first worked in the context of vine modeling as explained in the next paragraph: our goal is to develop a fully automatic method, unlike the methods proposed in the related work. Moreover, we can not rely on the assumption of having visible branches (see for example Figure 2.5).

2.2.2.1 The context

This work was done in the context of the projetc VINNEO which goal was to propose innovating techniques in order to improve the quality of the local wine[2]. Our participation aimed at characterizing vine properties based on images. Differences between different vine plots, or within a vine plot, should be identify for clustering the grapes into different products (wine). For example, an important factor is the SECV (*Surface Exposée du Courvert Végétal* in french, Exposed surface of the foliage). It corresponds to the total surface of all the leaves of a vine. Studies have shown, that to properly mature a kilogram of grapes, the vine needs from 0.5 to 2 square meters of SECV depending on the species, weather conditions or the type of wine

[2] `http://www.inp-toulouse.fr/fr/partenaires/innover-avec-l-inp-toulouse/toute-l-actualite--partenariats-et-transfert/l-inp-toulouse-impliquee-dans-vinneo.html`

produced. The ratio between leaves and fruits is calculated by dividing the SECV by the plot performance. It is therefore useful for winemakers to estimate the SECV on different plots in order to define the number of clusters/products to consider on a given vine plot. Today, this index is just estimated by eye, the winemaker simply looking at the wines. We thus proposed to embed a camera on the tractor in order to automatically extract the SECV from the images. As an intermediate step, we reconstruct a 3D vine plant for these images. An automatic method segments the foliage, based on the assumption that the camera motion is linear and parallel to the wines (more details are given in [Guénard 2013a]). Then, the images are rectified, to get a fronto parallel view of the plants. The resulting image is therefore of limited quality.
So, the method we developed is adapted to this context: we assume that we know the species of the plant, wines, and benefit from the fact that the vines grown in a plane (because of the trellising). The fist step of the proposed method consists in extracting a branching structure from the foliage segmentation in a 2D image. The goal of this method is to extract a possible skeleton, from which a plant is generated, and its projection is then compared to the original binary shape.

A proof of concept

First, we wanted to assess that it is indeed possible to estimate a skeleton from the binary shape of the foliage, with biological knowledge on the plant. We asked two wine experts[3] to draw the structure of the branches on images of vines rectified in the plane where they grow – We assume that the entire structure of the plant is in the same plane as the branches are attached on parallel iron wires by the winemakers. The resulting drawings from the two wine experts are shown in Figure 2.5. First, their knowledge on the plant growth and the human interaction (pruning and

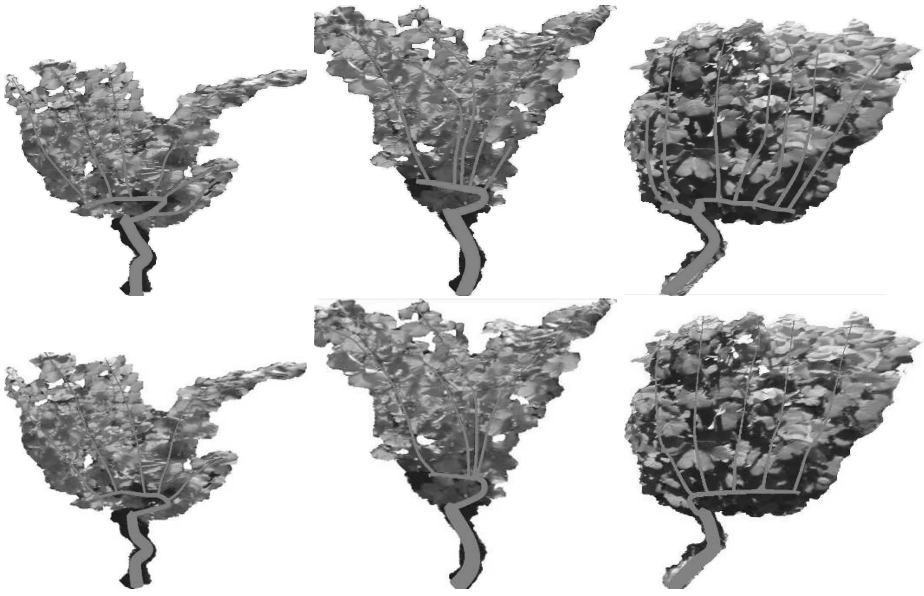


Figure 2.5: Example skeletons estimated by two different wine experts on three different vine plants.

trellising), together with the visible foliage information allowed them to quite quickly and easily propose a skeleton for the plant, and, as a matter of fact to guess the type of pruning of the

[3]Eric Serrano, Regional Director of the French Institute of Vine and Wine southwest and wine and engineer Jean Hemmi, Sub Director of Vinovalie group responsible for the cellar of Fronton

mother branch –the branch attached directly to the trunk. Moreover, the experts found in all cases (15 plants) very similar mother branch shapes, a consistent number of branches (equal up to one or maximum two branches), and coherent branches shapes. So, together with their knowledge on the plant, the shape of the foliage projection provides them enough information to estimate a relatively stable branching structure.
Thus, our goal is to automatize the process of finding branching structures similar to the branches traced by experts. We are naturally interested in the skeletonization of a binary form and we sought to develop a method such that it is possible to take into account a prior knowledge related to biological constraints.

Skeleton extraction: related work

Skeleton curves are used in many applications in 2D and 3D. In particular, they may be used for giving a simple and intuitive representation of the object, and as such are useful for editing or animating objects. In our setting of plant modeling, the goal is to extract the curve structure, that is, its branching system. The most classical skeleton is the medial representation of a 2D shape and has multiple definition. Intuitively, the medial axis is the locus of points located in the middle of the shape; each point has a corresponding local thickness. Blum [Blum 1967] first defined the medial axis as the set of the centers of maximal balls. A ball is maximal if it is included in the shape, and is not contained in any other ball included in the shape (see Figure 2.6 *left*). An alternative definition is to define the medial axis as the locus of the set of points having at least two closest points on the boundary (see Figure 2.6 *middle*). Amenta et al. [Amenta 1998] showed that for C^1 boundaries, the sphere centered on the medial axis and passing through the two closest point is tangent to the boundary. Finally, assuming a fire propagates from the boundary to the interior of the shape, the medial axis is also defined as the location where two or more burning fronts meet (see Figure 2.6 *right*). The main drawback

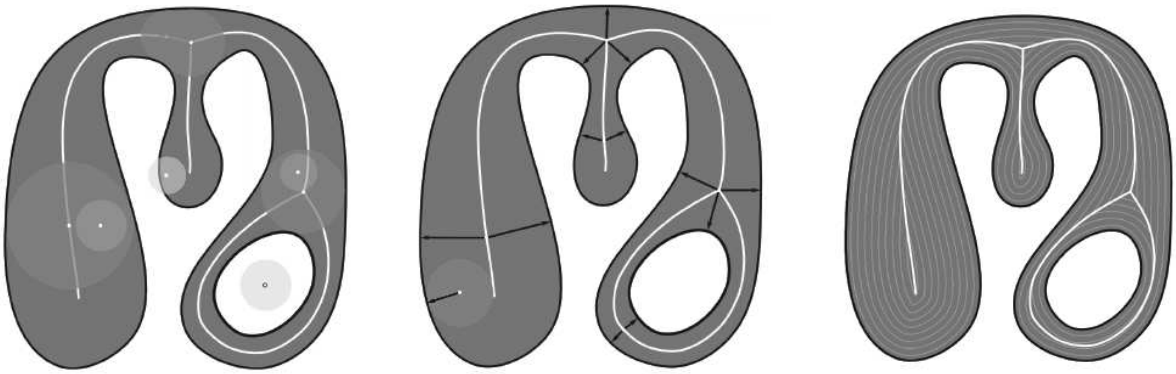


Figure 2.6: Three different definitions of the medial axis: centers of maximal balls (*left*), points having at least two closest neighbors on the boundary (*middle*), and shock graph of a grass-fire evolution from the boundary (*right*) (images from [Tagliasacchi 2012]).

of medial axis skeletons is their lack of robustness. Many authors have proposed approaches to prune the medial axis in order to keep important or stable parts (see [Attali 2009] for a survey). In [Shlyakhter 2001], the authors propose to use the medial axis to generate a branching structure of a tree. When the leaves are added to this branching structure, the tree with foliage has a satisfying shape. However, the branching structure itself is not realistic: it look very different than a natural branching system, in particular since branches are not growing up but in any

direction. Indeed, many properties of the medial axis are useless for our setting. For example, the medial axis preserve the homotopy type of the shape. In the case of a binary shape modeling the projection of the foliage, holes may appear but do not necessarily require an additional branch to be generated. Also, a branch/skeleton does not need to be centered. For that reason, we propose to adapt the skeleton extraction method of Cornea [Cornea 2005] (a comparison of classical methods for skeleton extraction and their respective properties is given by Cornea et al. [Cornea 2007]). Cornea et al.'s method produces a skeleton which is not necessarily homotopic to the shape, neither centered. However, the skeleton is connected, robust (stable to small changes), and smooth. This skeleton will define the branches of the tree.

2.2.2.2 The proposed method

Cornea's original method

We start with a discrete binary shape corresponding to the segmented foliage projection, that is, a connected set of pixels. We consider 8-connectivity, that is, any pixel has eight direct neighbors. Cornea's original method [Cornea 2005] takes two steps (Figure 2.7):

- First, we define a vector field on the shape. Each pixel $\mathbf{p_i}$ of the binary shape $\mathcal{B}$ is associated a vector $\mathbf{p_i}$, computed as the weighted average of the pixels m_i on the boundary Ω:

$$\overrightarrow{\mathbf{f}_i} = \sum_{\mathbf{m_j} \in \Omega} \frac{1}{||\overrightarrow{\mathbf{m_j p_i}}||^2} \frac{\overrightarrow{\mathbf{m_j p_i}}}{||\overrightarrow{\mathbf{m_j p_i}}||}. \tag{2.1}$$

- Second, the inner pixels p_i such that $\mathbf{p_i}$ is closed to zero are defined as *critical points*. The skeleton consists in the paths from the critical points following iteratively the vector field.

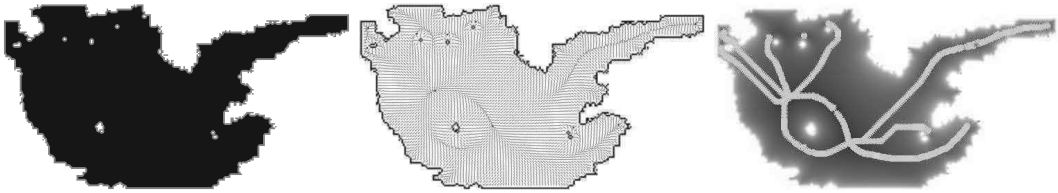


Figure 2.7: Cornea's original algorithm on a binary shape corresponding to a vine foliage (a). The boundary Ω appears in red. (b) The vector field computed on the binary shape by equation 2.1), and the corresponding critical points in blue. (c) The extracted skeleton, in green.

Adapting the skeletonization for a branching system

To extract a skeleton of the binary shape modeling a branching structure, we need to inject some domain/botanical knowledge. First, we want to enforce some anisotropy, since branches grow up. Second, we want to have more than one branch in large area. Our idea is to artificially create contour points within the binary shape, in order to partition the skeleton, as illustrated in Figure 2.8

We want to partition the segmented foliage in areas potentially covered by one branch. For that, we define some so called *cuts* in the binary shape. This segmentation is done following some botanical knowledge on the branches shape (like the angle with the trunk) and the shape of the foliage. As we aim to produce multiple models and select the best one, the parameters for the cuts are random variables.

Figure 2.8: The skeleton extracted from Cornea's original method, using the red boundary (left). If we artificially partition the initial shape adding some boundary point, the skeleton will respect these boundary: our goal is to partition the shape in ordre to avoid vertically large areas.

Partitioning the binary shape

The cuts are defined iteratively, the starting points are chosen based on biological assumption on the plant species. For example, for monopodial plants, that is, plant whose branching system is organized around a central trunk, we give the interval between successive branches, as well as the angle between the branches. For vines, the number of branches, and therefore the number of cuts, is also defined as an input of the algorithm. For finding the ending point of the cut, a DCE (discrete curve evolution) algorithm [Latecki 1999] is applied on the boundary Ω of the plant in order to characterize and favor the inward angles of the shape as illustrated in Figure 2.9.

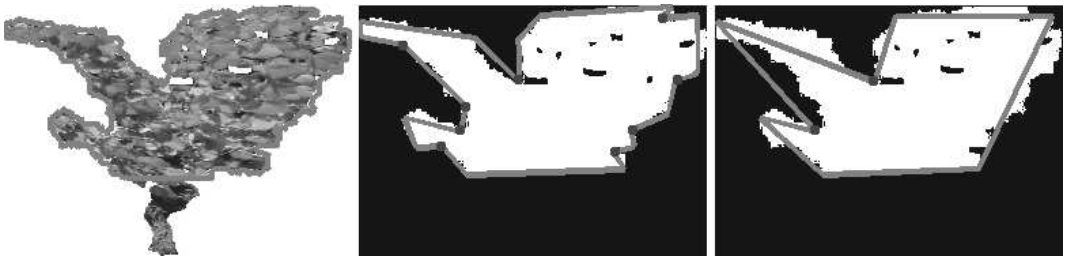


Figure 2.9: On the left, the foliage and its boundary Ω. In the middle, and right, the DCE curve with respectively 14 and 8 vertices. Vertices of inward angle $\leq \pi$ are shown in blue: they most likely indicate the limit between two branches regions.

Cuts are then chosen iteratively among DCE vertices, following a density of probability to favor inward angle. A filter avoids taking cuts too close to each others, or crossing. Each given cut is associated a probability which is the used as prior knowledge 2.2.4. More details can be found in [Guénard 2013a]. Figure 2.10 shows two examples of cuts, one for vines and one for a monopodial plant.

Probability map on the binary shape

The cuts are not given directly to the skeletonisation algorithm; we first apply a horizontal smoothing (one dimensional Gaussian filtering) to avoid branches to follow too closely the shape of the cuts. Figure 2.11 shows an example of the probability map corresponding to the cuts of Figure 2.10. Then, we generalize Cornea's vector field computation to take into account all

Figure 2.10: On the left, cuts for 5 branches on a vine. On the right, cuts for a monopodial plants (the cuts are cubic curves so a end point and a tangent is given on each side of the cut.

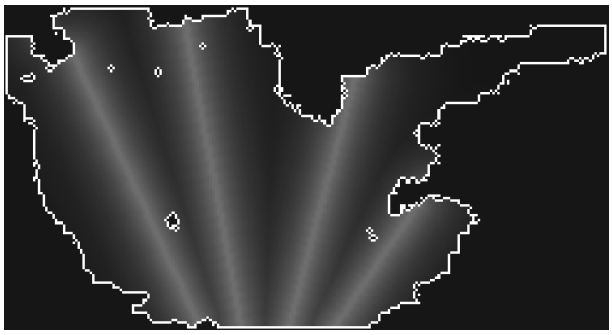


Figure 2.11: Probability map $\mathcal{P}$ corresponding to the cuts of Figure 2.10 *left*.

interior points with a non zero probability to be a contour point:

$$\overrightarrow{\mathbf{f}_i} = \sum_{\mathbf{m_j}\in\Omega} \frac{1}{||\overrightarrow{\mathbf{m_j p_i}}||^2} \frac{\overrightarrow{\mathbf{m_j p_i}}}{||\overrightarrow{\mathbf{m_j p_i}}||} + \sum_{\substack{\mathbf{p_j}\in\mathcal{B}\setminus\Omega \\ j\neq i}} \frac{\mathcal{P}_j}{||\overrightarrow{\mathbf{p_j p_i}}||^2} \frac{\overrightarrow{\mathbf{p_j p_i}}}{||\overrightarrow{\mathbf{p_j p_i}}||} \tag{2.2}$$

Figure 2.12 shows the resulting vector field.

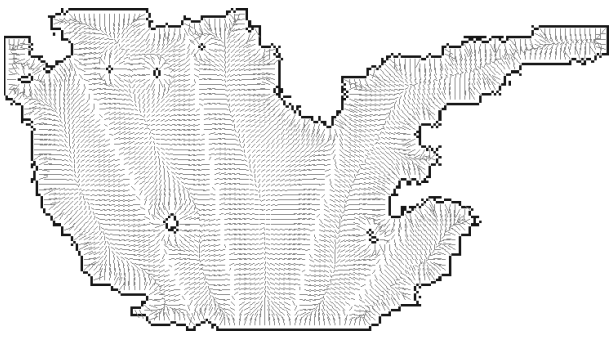


Figure 2.12: The vector field computed from the probabilty map on Figure 2.11 using equation 2.2.

The last step of Cornea's algorithm is applied to define the skeleton, or set of attracting points that will be used to fit the branching system (Figure 2.13) .

The branching system

We now need to fit a parametric model of the branching system with the attracting points describing the skeleton of the binary shape. The parametric model has been created in L-py [Boudon 2010] and each branch is modeled by a degree 3 Catmull-Rom curve [Catmull 1974] . The control points are computed in order to fit the parametric model and the attracting points using (in a least square sens). Figure 2.13 shows the resulting branching system for the vine. Note that the resulting skeleton branches give a realistic branch structure. Moreover, they not only fill the space in a way coherent with the boundary shape, but also avoid the holes in the binary shape.

Figure 2.13: The skeleton computed from the vector field 2.12.

Hierarchical branching system

For more complicated plants, the partitioning is used iteratively, as illustrated by Figure 2.14 .

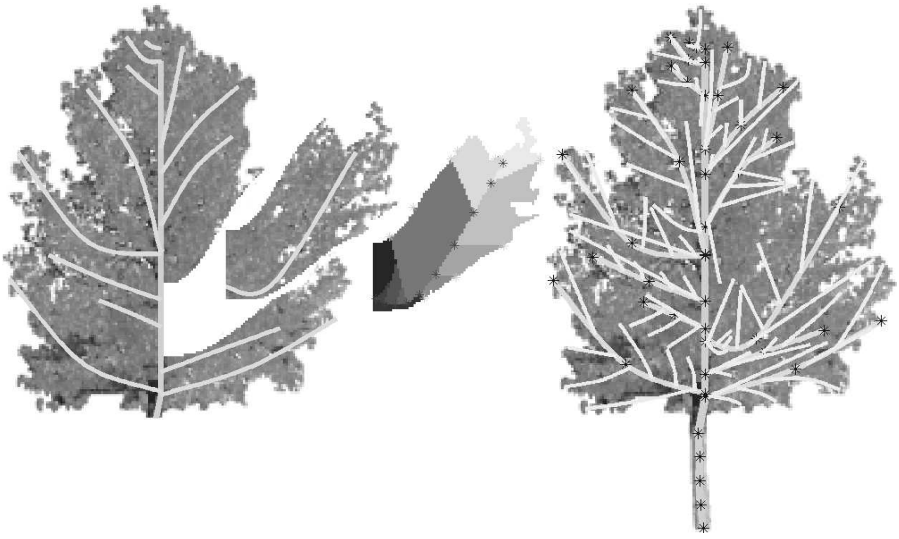


Figure 2.14: On the left, the first order branches –in cyan– (attached to the trunk –in green–) are computed using our approach. Then, on each cell (an area between cuts), the algorithm is applied recursively to find second order branches. On the right, second order branches appear in yellow.

2.2.3 Generating a 3D plant model

From the branching system in 2D, we now want to define a real plant in 3D. For that, we first need to infer a 3D plant from the 2D branching system extracted in the previous session. Then, branches are added a thickness and texture, and more importantly leaves are added to the model. Finally, we use a selection criteria measuring the posterior probability for choosing among several possible models.

2.2.3.1 A 3D skeleton

For the vines examples, the 2D skeleton is general enough since main branches are attached in a same plane. However, in general trees grow in 3D! We treat here the case of monopodial trees, and follow the work of [Zeng 2006] and [Okabe 2005].
First, we want to maintain the correspondence between the 2D binary shape and the 3D model, that is, we require that the projection of the 3D tree still fits the 2D segmented foliage. For that, we define a 3D volume in which the branches will grow: each horizontal segment of the 3D shape is associated a 3D horizontal circle. Together, these circles define a volume (Figure 2.15). In order to avoid the 3D volume to be biased in the direction of the foliage projection, the depth of the circle centers follows a Gaussian density centered on 0.

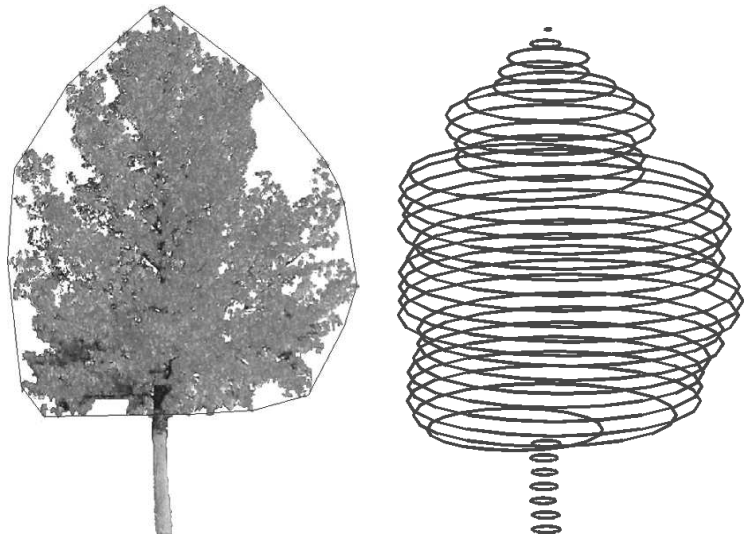


Figure 2.15: On the left, the original image of the liquidambar (sweetgum) with a segmentation of its foliage in 2D. On the right, the considered envelop for the foliage considered in 3D.

Each branch that does not reach the foliage boundary Ω is turned so that it reaches the volume boundary. However, the branch could be going towards the front or the back. Following Okabe et al's [Okabe 2005], the branches are spread as to maximize the angle between two branches from a top view. Also, branches need to be added to get a tree with a good density: several skeletons are computed and eroded to generate enough branches (Figure 2.16).
Figure 2.17 shows an example of 3D skeleton extracted from our method.

2.2.3.2 Texture and leaves

The branching system is then represented with a thickness and texture. Textures from the tree species are used. The radius of the branch is a function of the distance of the branch to the trunk and the order of the branch. Parameters for the leaves are extracted on the 2D skeleton:

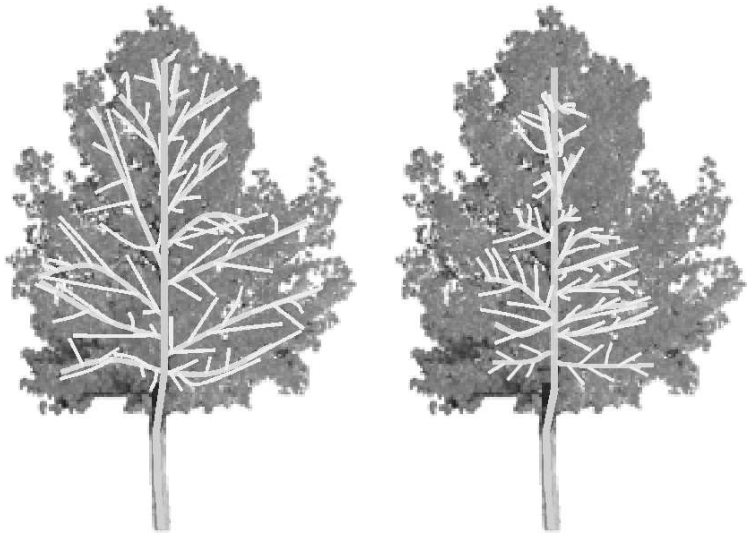


Figure 2.16: On the left, the skeleton after a small erosion. On the right, the skeleton after a larger erosion.

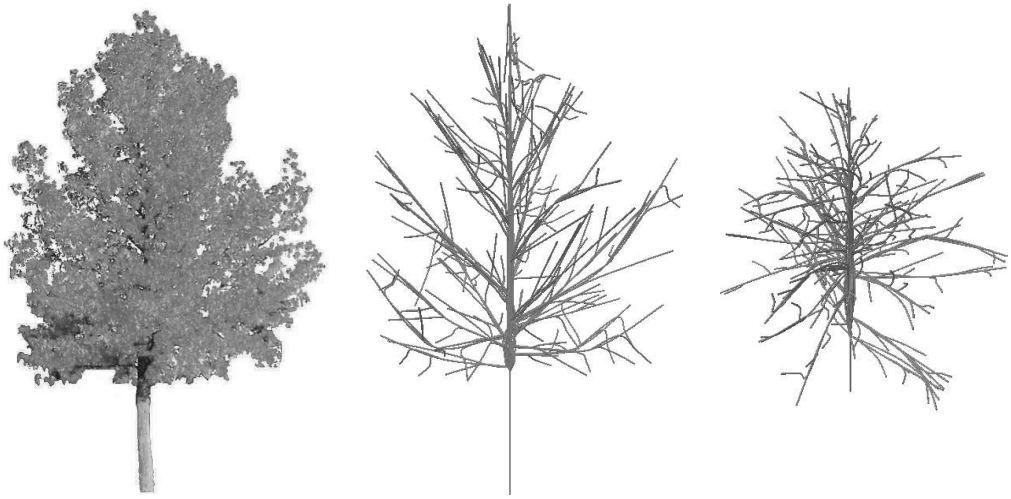


Figure 2.17: On the left, the input image for the liquidambar. In the middle, the complete skeleton seen from the same viewpoint. On the right, the same skeleton seen from another viewpoint.

- a **leave radius** (Figure 2.18) is attached to each point of the branch, as the length of the segment on the line perpendicular to the tangent of the branch, and included in the cell of the branch (the cell is the area between cuts);

- a **density factor** corresponding to the percentage a foliage pixels in the cell is also attached to the branch.

On the generated models, the leave parameters are random variables following a Gaussian law whose mean is the computed value. Figure 4.16 illustrate the use of the leave radius and density influence.

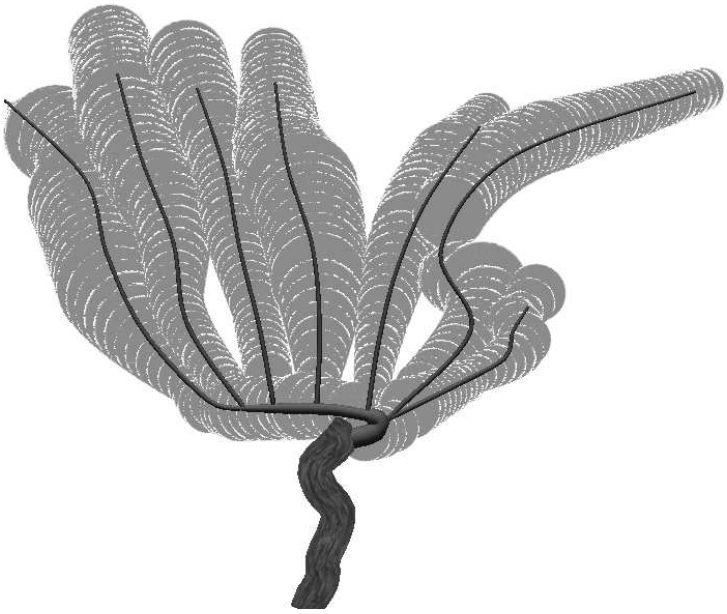


Figure 2.18: On each branch, circles illustrating the computed leaves radius.

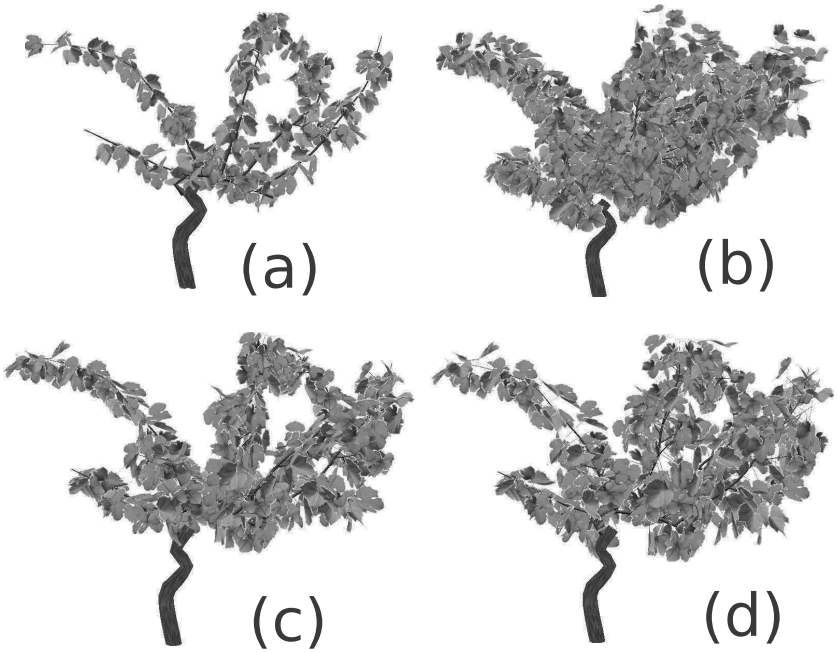


Figure 2.19: The same branching system with varying parameters for the leave radius r_f and density d_f: (a) r_f and d_f small (a), (b) r_f and d_f large, (c) r_f small and d_f large, and (d) r_f large and d_f small.

2.2.4 Model selection: a Bayesian approach

So far, we have proposed the construction of a model of a tree, following density functions depending on both knowledge of the species of the tree, and on the shape of the foliage. Some prior knowledge can also be added: in the case of vines, for example, we can give a probability for the shape of the main branch depending on the region where the vine grows. Also, we have observed the number of branches in the vineyards where we went: the average number of

branches left on each foot follows a normal distribution with mean 5 and standard deviation 2.

The model selection process is done by maximizing the posterior probability, following a Bayesian approach. If we call $\mathcal{M}$ the model, and $p(\mathcal{M})$ the probability of an instance of the model. We want to maximize the posterior probability $p((\mathcal{M})|\mathcal{I})$, that is, the instance of the model maximizing the probability of the model given the image. Baye's rule says:

$$P((\mathcal{M})|\mathcal{I}) \sim \mathcal{P}(\mathcal{M})\ \mathcal{P}(\mathcal{I}|\mathcal{M}).$$

Here we still need then to evaluate the likelihood $P(I|\mathcal{M})$. We first project the model of the plant $\mathcal{M}$ with the similar viewpoint as the view point from the image during acquisition: assuming that the principal point is the center of the image, we estimate a theoretical position of the point of view that is on a line orthogonal to the image plane and passing through the principal point. So then transform the model image into a binary image $\mathcal{I}$ (1 if there is foliage or branches and 0 otherwise). This binary image is then compared to $\mathcal{B}$ (see section 2.2.2.2) which corresponds to the binary image of the original image (1 if there is foliage or branches and 0 for background) . The likelihood is:

$$P(\mathcal{M}|I) = 1 - \frac{\sum_j (\mathcal{I}_j - \mathcal{B}_j)^2}{\#pixels}.$$

where $\mathcal{I}_j$ and $\mathcal{B}_j$ are respectively the jth pixel of $\mathcal{I}$ and $\mathcal{B}$, and $\#$ *pixels* corresponds to the number of pixels in the bounding box of the two images.

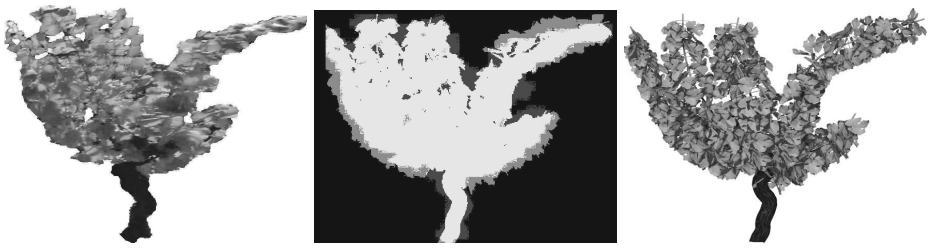


Figure 2.20: On the left, the original image of a vine plant; on the right the projection of a model generated by our method; in the middle, the error map between thiese two binary images. The grey pixels correspond to pixels were the two images do not match.

We sample the set of models, and select among the proposed model the model maximizing the posterior probability (Figure 2.21).

2.2.5 Results

First, we give here visual result for generated monopodial trees from a single image (Figures 2.22, 2.23 and 2.24).

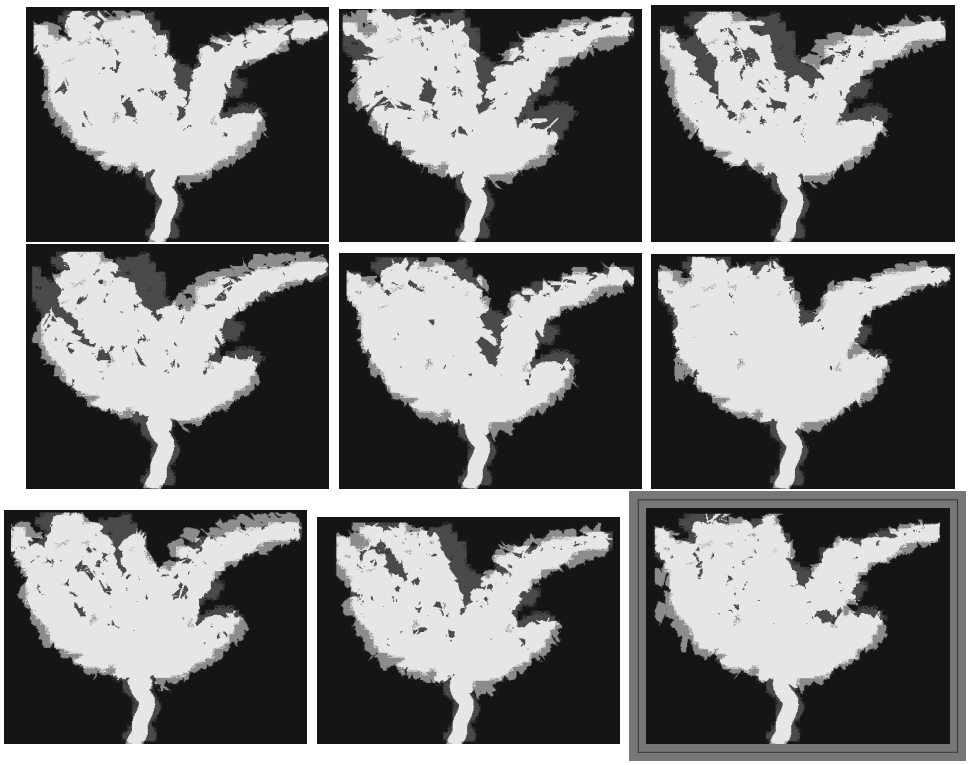


Figure 2.21: The selected model (with the red box) maximizes the posterior probability.

Figure 2.22: The liquidambar (monopodial tree). On the left the original image, in the middle, the generated model rendered from a similar view point; on the right, the same model rendered from a different viewpoint.

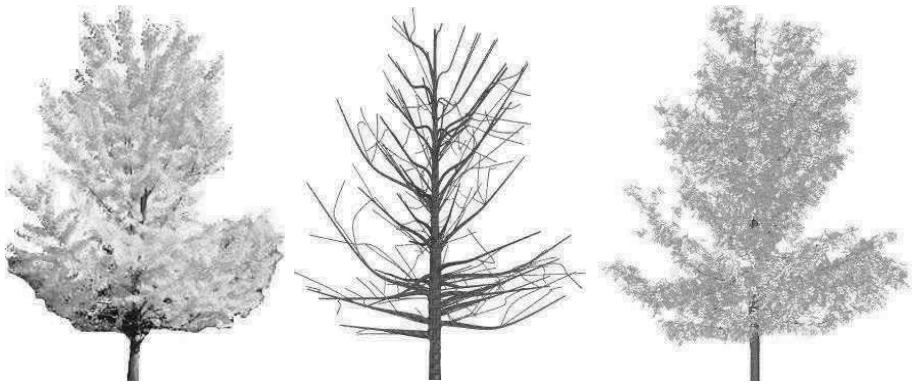


Figure 2.23: The ginkgo biloba (monopoldial tree). On the left, the original image; in the middle, the generated branching system (3D); on the right, the generated model with leaves, rendered with a similar viewpoint as the original image.

Figure 2.24: The fir. On the left the image of the fir, and one the right the generated model from a similar view point.

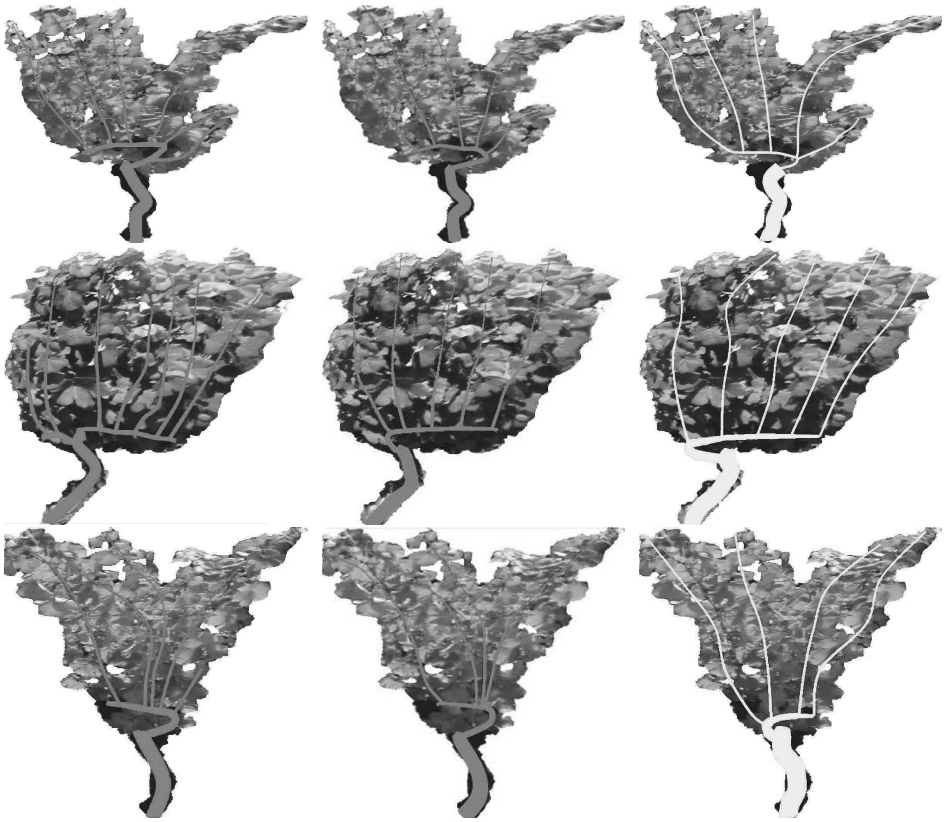


Figure 2.25: Comparison of the branching systems found by the experts (in red) and the branching system found by our method (in yellow). Note the when the shape of the foliage is very compact, our method tends to fill it with more uniformly spaced branches than experts.

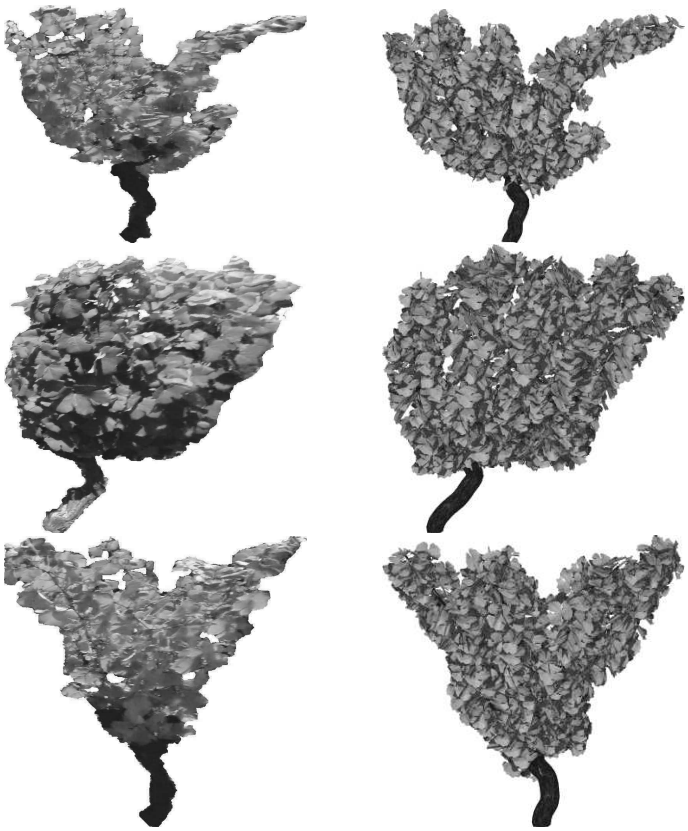


Figure 2.26: Three examples of reconstructed vines plants. On the left the original images; on the right, the reconstructed models rendered with a similar view point.

For the vines, we also give a visual result, and a comparison with the experts branching systems (Figure 2.25). We see that we were successful is automatizing the process of deducing the branching system from the image. Finally, we give some example of reconstructed vines (Figure 2.26). More quantitative evaluation for the vine case can be found in [Guénard 2013a].

2.2.6 Conclusions, limitations and perspectives

As shown by the comparison with experts (Figure 2.25), we have met the goal of mimicking the extrapolation of the structure of the tree (its branching system) from a single image. For that, we have included the knowledge of the experts in the 3D generated model, and instantiated parameters of the model by analyzing the image of the foliage, and selecting, using a Bayesian criteria, among numerous proposed instances. In that sense, our approach may be compared the analysis-by-synthesis approaches (e.g. [Yuille 2006, Nair 2008]).

One of the direct limitation of our approach is not to extend to general trees. We did try our method on regular tree (non monopodial) but the result are not satisfying yet (Figure 2.27).

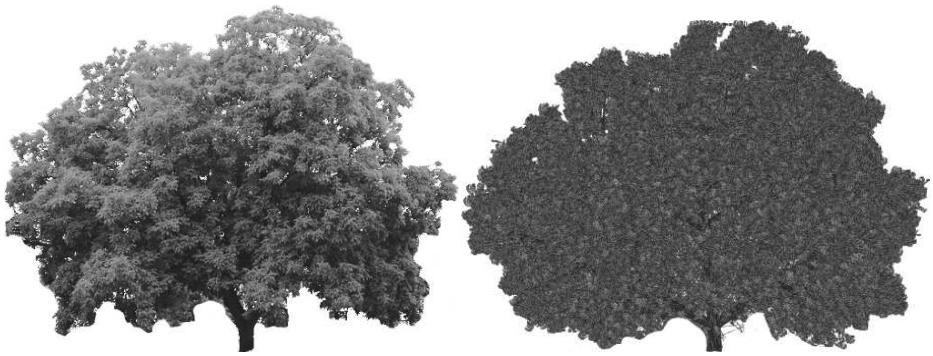


Figure 2.27: Example of a nut tree. On the left the original image, and on the right the recnstructed tree from our method rendered from a view point similar as in the original image.

We believe though that the best way to recover the irregular shape in the canopy would be to use the bump in the canopy to apply some generalization of the cuts into 3D (Figure 2.28.

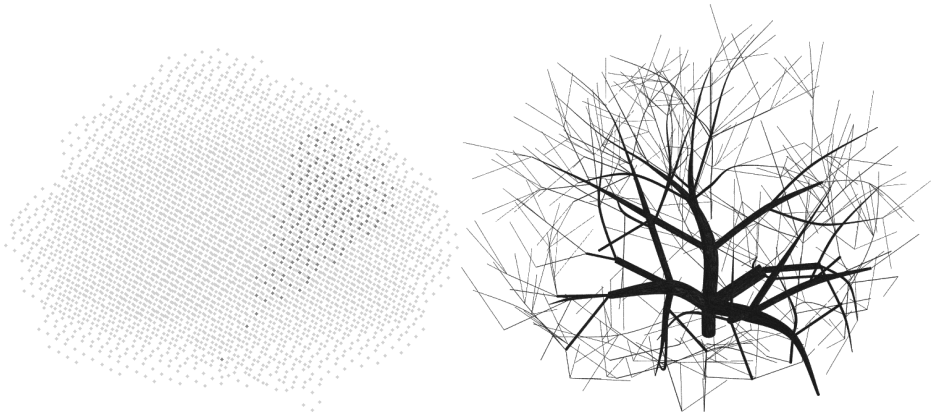


Figure 2.28: Example of a nut tree. First try for the generalization of the partitioning method in 3D; on the left, a 3D cut (in red); on the right, the generate branching system.

A more ambitious extension of this work would be to be able to somehow learn, or infer the knowledge we used on the model, what we called biological knowledge, from a large set of images of the same species. A requirement could be to have some images from trees in the winter as well, to get the properties of the branching system.

2.3 Generating 3D from one image: Spline from Shading

This next section succinctly present some results of a work done in a very different context but that share many characteristics with the previous framework; we also work with a single image, and the prior knowledge here is the chosen model: a B-spline tensor product, parametric surface.
Another way to reconstruct 3D from a single image is Shape from shading (SFS), which consists in reconstructing a functional surface from a single gray level image mapped on the domain [Horn 1989]. This technique had been considered rather like a theoretical exercise until a more realistic camera model was considered by three independent groups [Courteille 2004, Prados 2003, Tankus 2005]. Initially, several unknowns per pixels were considered, the depth value at each pixel and possibly some derivative of the depth function, which lead to a very large number of unknowns. Techniques to speed up the process [Szeliski 1991] have been considered, as well as multiresolution technique like a resolution on a grid [Lee 1996, Terzopoulos 1986].
A natural idea to limit the number of unknowns (and thus avoid the need for boundary conditions) is to consider a parametric model: [Pong 1989, Bora 1990, Saito 1994] have proposed to reduce the number of parameters considering piecewise linear, quadratic, or super-quadrics models. By fixing the 3D model, the author assume some knowledge on the solution. However, whereas piecewise linear models may lack regularity, quadric based model are not freeform. Natural models to consider are polynomial models [Kim 1997]. By considering polynomials in the Bersntein, or piecewise polynomials in the B-spline basis however, the partial derivative can expressed very easily [Courteille 2006a]. In the next section, we show the result of SFS when using a parametric tensor product B-spline model.

Reconstructing a functional tensor product B-spline surface

As mentioned, the prior knowledge used here is embedded into the model we consider: piecewise polynomial functional tensor product surface uniform B-spline basis with a given degree (3) and a chosen number of control points. In this particular setting, the B-spline model provides a series of appropriate properties. The reconstructed surface is clearly functional. The chosen parametric model reduces significantly the number of unknowns, whereas being freeform. Taking a tensor product is natural since derivatives in the two directions are very easy to compute: the derivatives are expressed linearly in terms of the control points. The control points (coefficients) of the derivative are the first differences of the control points of the function up to a constant factor (the degree in that direction). Moreover, B-splines have a inherent multiresolution structure. By choosing a degree 3 spline, the model is assumed to be smooth, but changing a control point applies only a local change on the surface, which is an important property for the search algorithm we use.

Results

The proposed solution is based on joint use of the B-spline model and a stochastic algorithm using simulated annealing (SA). The resolution for a coarse solution is computed using SA. The outcome of the resolution can easily be controlled in term of curvature (second order derivative of the B-spline function is readily available) and will be discarded if the sign of the curvature is not compliant (in the SFS resolution, the curvature of the reconstruction if defined up to the sign). Then, a deterministic approach refines the given solution.
Experiments on two $256 * 256$ images are shown for orthographic projection in Figure 2.29: in

column (a), a DEM[4] image simulated under orthographic projection, and a real image of a vase are shown; column (b) shows the corresponding shapes. The computed surfaces are represented in column (c) using our method, and in column (d) using Tsai and Shah's method [Tsai 1994]. We use $16*16$ control points for the DEM, and $9*9$ for the vase, which are the best compromises between a good reconstructed surface and a low computing time. Visually, our method produces more satisfactory results than Tsai and Shah's. For DEM, even if the surface contains several convex and concave areas, the reconstructed surface using the proposed method is very close to the real surface. The example of the vase shows that our method is does not suffer as much from noise: the smoothness of the underlying model insures robustness. Quantitative results shows that our method also performs better in terms of errors 2.30. We compute the two error estimators:

$$|\Delta u|_1 = \frac{1}{Card(\Omega)} \sum_{x \in Omega} |\tilde{u}(x) - u(x)| \tag{2.3}$$

$$|\Delta u|_\infty = max_{x \in Omega}\{|\tilde{u}(x) - u(x)|\} \tag{2.4}$$

where x is a pixel of the considered domain Ω, u is the ground truth, and $\tilde{u}$ the estimated function.

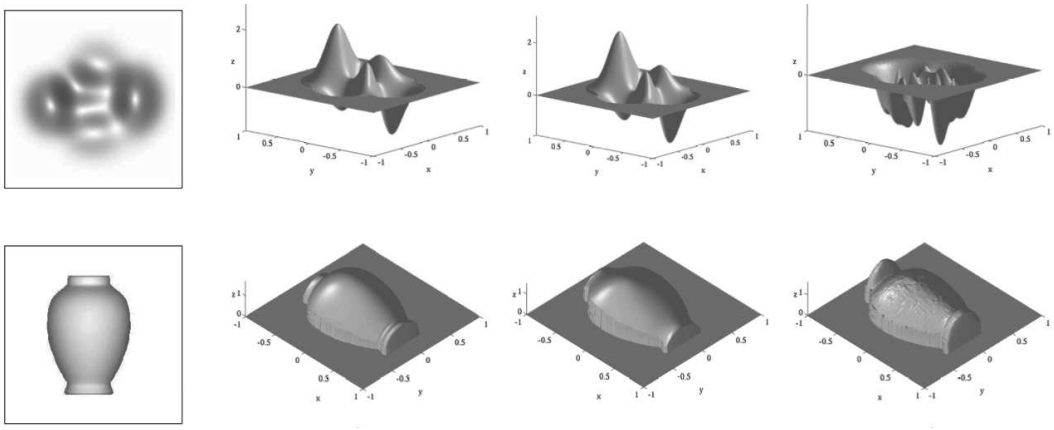


Figure 2.29: Under the assumption of orthogonal projection. The images (a), the ground truth (b), the proposed method (c), Tsai and Shah's result [Tsai 1994] (d).

	BS-SA	TS
$\|\Delta u\|_1$	$0.77e-01$	$3.61e-01$
$\|\Delta u\|_\infty$	$0.72e+00$	$2.73e+00$

(a)

	BS-SA	TS
$\|\Delta u\|_1$	$1.90e-01$	$3.30e-01$
$\|\Delta u\|_\infty$	$1.28e+00$	$1.49e+00$

(b)

Figure 2.30: **BS-SA** is the proposed method, and **TS** is Tsai and Shah's method on the DEM model (a) and on the vase (b).

[4]the function `peaks` from `Matlab`

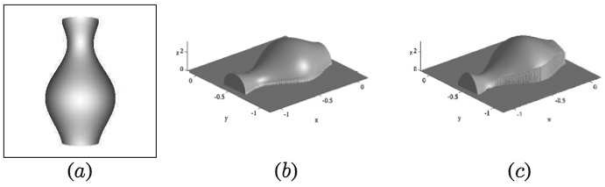


Figure 2.31: (a) Synthetic image of a vase simulated under perspective projection; (b) corresponding surface; (c) reconstructed surface using **BS-SA**.

Finally, Figure 2.32 represents a real image of the mouse, and the computed surface using our method, considering a domain Ω where pixels on the hand have been discarded.

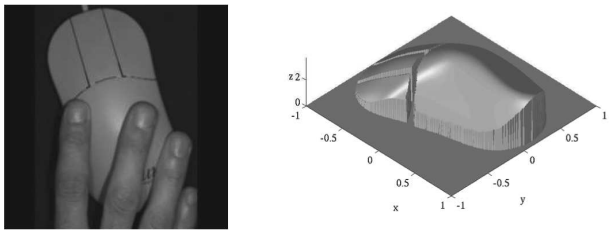


Figure 2.32: **BS-SA** is the proposed method, and **TS** is Tsai and Shah's method on the DEM model (a) and on the vase (b).

Figure 2.31 illustrate the result for perspective projection: (a) represents the synthetic image of the vase simulated under perspective projection, (b) the computed shape using our method where the camera has been calibrated (the intrinsic parameters are given).

2.4 Conclusion, limitations and perspectives

This section has presented two contributions for generating 3D models from images. In both cases, we determine the parameters of a chosen parametric model. Defining a parametric model adapted to the problem can be seen as defining some prior knowledge on the solution, assuming a particular shape for the tree modeling, or a degree of smoothness for the SFS surface reconstruction.

A possible extension of the proposed modeling from images is to model dynamic objects from image sequences, or videos. Of course, the generalization is not direct, but for trees, the branching system corresponding to a skeleton, the generalized cylinder representation is naturally adapted for animation. Some recent work [Li 2011] has proposed a dynamic modeling of a tree from a fixed viewpoint video sequence. In this context (tree modeling and fixed viewpoint), it is reasonable to assume that the chosen branching system on the first image can model both the tree on the first image and the subsequent motion. In a more general context, a 3D model able to evolve and adapt over time is probably necessary.

Because 3D reconstruction from a single image is inherently an under-constrained problem, the prior knowledge is necessary to restrict the solution space. However, working from an image is interesting. First, images are very easy to produce. Nowadays, almost any mobile phone is equipped with a camera. Moreover, the image of an object gives also a lot of information on the appearance of the object. Nevertheless, inferring 3D information from a single 2D image is an easy and natural task for people, but a very hard and challenging task to automatize. So, to help getting this missing semantic interpretation, one way is to rely on user interactions. As an example, [Chen 2013] proposes an approach for easily generating (and editing) 3D models from a single image. They also consider prior knowledge on the 3D shape assuming the reconstructed surfaces are sweep surfaces. The shape of the section curve is drawn by the user, and the path followed by the section is also roughly indicated by the user, and as well guided by the edges of the image. A reconstruction of the textured object is done very effectively (interactive time). Then, a direct edition of the 3D model can be done, by changing the parameters of the shape, moving parts around, or copy/pasting them. Sketching is another research area seeking for easily and intuitively modeling 3D (see [Olsen 2009] for a survey), targeting modeling for non-expert users. Sketching has been used as a stand alone approach, but also together with images [Thorne 2007]. However, both the interactions mentioned in Chen et al. work and in Sketching are explicit. A very challenging problem would be to implicitly deduce 3D information the user can infer in an image from user interactions, that is, using crowdsourcing techniques.

Chapter 3

Manipulation of 3D content: Object Tracking and Analysis

Contents

3.1 Introduction

In order to use, re-use, or present 3D models, we need good tools for their manipulation. In this chapter we present three different contributions. These contributions are quite different; they work on different models: the first one considers point based models, the second one parametric surfaces, and the last one textured meshes. They also strive different solutions. The two first contributions present methods that form the basis for further developments. The first part is proposing a tracking algorithm, where a 3D model of an object is appearing in a video.

Video tracking have many applications, like in security and surveillance (e.g. traffic control), or video editing. Here, since the 3D nature of the object is known, recovering the pose of the camera opens the possibility for augmenting the video. Like the two methods for generating 3D models presented in the first part, we start here from 2D content. In the second contribution, we analyze 3D models, to find intra-model similarity. Here also, there are many applications, like compression, edition of objects, shape indexation. The last contribution concerns the navigation when inspecting a 3D objects. Our experiments show that 3D objects are not easy to manipulate (navigating around a 3D object is too slow). It is therefore important to develop tools to simplify accessing 3D objects, editing 3D content, in order to provide good support for their use in multimedia applications.

The work presented in this chapter has been developed in the context of the doctoral work of Christophe Dehais (tracking), Viet Dang Quoc (similarity detection) and Phuong Nghiem (3D interactions). For the first part (section 3.2), more details can be found in Christophe's Dehais Ph.D.[Dehais 2008], co-advised with Vincent Charvillat. The contributions have been published in [Dehais 2006, Dehais 2010]. The second part is the current work of Viet Dang Quoc, and has been published in [Dang 2012, Dang 2013]. This work is co-advised by Sandrine Mouysset whose expertise is in classification. The part on interaction is developed in the context of the doctoral degree of Phuong Nghiem, and has been published in [Nghiem 2012, Nghiem 2013]. Axel Carlier (another Ph.D. student of the group) have also been taking part into the work. This work is co-advised with Vincent Charvillat.

3.2 Object tracking

In this first section, we introduce the use of point-based 3D models for real-time visual object tracking with a single monocular camera. Previously, this problem as been addressed using sparse 3D models based on edges, meshes or textured patches. Instead we use a point-based model and related methods developed in the computer graphics community. The points are arbitrarily sampled on the object surface and no connectivity information is required. We show that state-of-the-art techniques for real-time rendering of point-based geometries can be efficiently recycled to be used in a 3D tracking context. We derive an original tracking algorithm from the method proposed earlier by Vacchetti et al. which combines an iterative pose update and a keyframe-based 3D registration. We first propose a mathematically sound framework for using point-based models for the purposed of visual tracking. This framework allows the reconstruction of dense linear motion predictors and the generation of novel views from keyframes for wide baseline feature matching. Both make use of the same general surface splatting technique, which we implement, together with other low-level vision tasks, on the GPU, leading to a real-time tracking algorithm.

3.2.1 Introduction

Many 3D object tracking algorithms run in real time under specific assumptions, that generally relate to the object itself, its motion (possibly including deformations) and the viewing conditions (illumination, geometric model of the camera, etc.). The choice of the object model in a visual tracking system will therefore constrain what can and what cannot be tracked. The goal of this paper is to propose a conceptually simple yet promising model for the object to track. While not dense point-based models and related GPU based techniques have been quite well studied

in computer graphics community, to our knowledge they have never been used in the context of visual tracking. In the following we briefly review the approaches classically proposed in the model-based 3D tracking literature.

The most common object models are based on wireframe 3D meshes which naturally lead to edge-based tracking techniques. The tracking problem is solved by registering 2D image features with the 3D polygonal mesh. In his pioneering work, Lowe [Lowe 1992] extracts image lines that are matched and fitted to those of the model. One can avoid preliminary edges extraction by actively searching for strong image gradient along the normal of the projected model edges [Kollnig 1997]. The measurements may be limited to control points sampled along the edges [Drummond 2002]. The model pose is then estimated by minimizing the distances between the control points and the detected image contours. These techniques are very well suited to industrial objects that show strong straight edges, but may fail for natural and smooth objects, for which the texture information may be more prominent.

To enrich the model with object appearance information, a common approach is to apply textures to the model surface. Those are generally aligned on the model thanks to real object images. Pressigout and Marchand [Pressigout 2007] fuse a model-based approach using edges with texture registration, yielding to the minimization of a single non-linear cost function. In the Active Appearance Models approaches [Xiao 2004] a compact texture representation is derived from the analysis of the dense object appearance in a (possibly large) number of poses.

Some objects (like faces) do not show strong texture information everywhere on their surface which makes the use of uniform texture information useless and even problematic because some parts of the model will lead to weak motion clues if any. Sparse texture representations using for example interest points help to address this issue. Vacchetti and al. show how 2D-2D correspondences between interest points provide 2D-3D correspondences between images and the object model registered along a set of keyframes [Vacchetti 2004]. Other approaches also integrate interest points correspondences and 3D constraints while tracking multi-planar structures [Simon 2002].

Bringing the idea of models based on sparse salient features further yields to collections of visual features without explicit topology. Several proposals use textured patches as a model for recognition or tracking tasks. The hyperpatches of Wiles et al. [Wiles 2001] are attached to 3D points and centered on corner-like regions in the image. Rothganger et al. [Rothganger 2006] introduced a similar 3D object representation using texture descriptors and applied it successfully to object recognition. The local patches also provide a compact representation of the object. Munoz et al. [Munoz 2005] use a set of small planar textured patches, a set of shape basis which encode the modes of deformation and a set of texture bases which represent appearance changes due to varying illumination.

We use a model similar in nature, made of a collection of unconnected points. But where Wiles suggests that a sparse representation of carefully located features is sufficient, we think that a dense cloud of points unrelated to image features is simpler to provide and set up. Point-based models are popular in computer graphics [Gross 2007] but have not yet made their way into the computer vision community. Points clouds are of particular interest since they are the typical output of most 3D scanning devices and the initial structure recovered by most vision-based reconstruction techniques.

In the following we show that efficient rendering algorithms designed for points clouds can be adapted and recycle for visual tracking. Thanks to an efficient implementation on recent GPUs, point-based models are handled in real time. Their high density and the flexibility allowed by the lack of explicit topology also form promising assets to manage changes in shape and appearance.

In the next section, we introduce our point-based model and the surface reconstruction framework used for both rendering and interpolating the surface attributes. Section 3.2.3 presents the tracking framework, based on an iterative pose update stabilized by using keyframes. Section 3.2.4 points out our main contributions: the reconstruction of linear motion predictors and the generation of new views from the keyframes. Finally, experimental results (section 3.2.5) highlight the benefits of this hybrid tracking approach and its efficiency for real-time applications.

3.2.2 Modeling and Rendering Point-based 3D Models

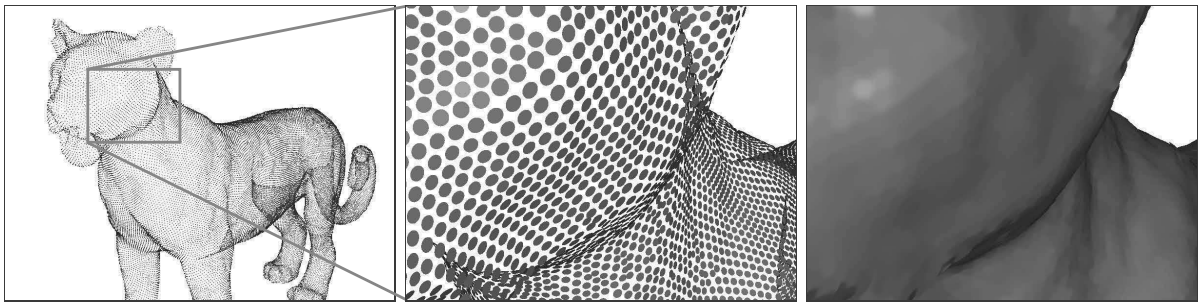
Point-based 3D models have become popular in computer graphics for both modeling and rendering purposes. There are two primary reasons. Firstly, 3D scanning devices typically output unstructured sets of points and meshing such data is a tedious task. Secondly, more complex editing tools and powerful workstations allows the creation of increasingly detailed models, while the final image resolution does not increase generally as fast. As a matter of fact when using triangles as the basic geometric primitive, their size tend to decrease to less than a pixel, rendering the setup and rasterization operations unnecessary.

3.2.2.1 Point-based Modeling

A point-based model defines a 3D object by a set of points sampling its surface, as depicted in figure 3.1 (a). No explicit connectivity information is necessary and the surface is not required to be regularly sampled. A fundamental issue for point-based graphics is the reconstruction of a hole-free continuous object surface from the samples (see figures 3.1 (b)-(c)). Point Set Surfaces [Alexa 2003, Levin 2003] are a popular meshless representations, yet they involve a costly reconstruction of a local surface approximation in object space and therefore are not well suited for real time processing. Other approaches involve rendering a "thick" primitive at each point in the image so that the holes are covered. Such methods date back to the QSplat algorithm [Rusinkiewicz 2000], where a hierarchy of circular footprints is used. Zwicker et al. [Zwicker 2001] formalized this approach by introducing the surface splatting algorithm. Each sample (or splat) is associated with a number of attributes (position, normal, color, etc.) and the rendering can be seen as a resampling process of these attributes on the regular pixel grid in image space.
Since this later algorithm is the main component of our tracking system, we detail it in the following section.

3.2.2.2 The Surface Splatting Algorithm

A splat $\mathbf{p}_i$ is defined by a 3D point, an oriented surface normal and a radius. When only a pure points cloud is available, the two latter attributes can be estimated by statistical analysis of local neighborhoods [Pauly 2003b]. The surface is further defined by a set of attributes $\{A_1^i, A_2^i, \ldots\}$. An attribute A is locally approximated using a function $\mathcal{P}_i^A : \mathbb{R}^2 \rightarrow \mathbb{R}^{n_A}$ defined in the splat plane (n_A is the dimension of the attribute space, e.g. $n_A = 1$ when A is scalar).
A radial (3D) reconstruction kernel r_i depending solely on the radius of the splat is defined in the splat reference frame. A typical choice is a gaussian kernel with a standard deviation adapted to the splat radius. Reconstructing the surface in image space involves the projection of points $\mathbf{y}_i$ from the reference plane of splat $\mathbf{p}_i$ to an image point $\mathbf{x}$. Let this projection be approximated by the linear mapping $\mathcal{M}_i : \mathbf{y}_i \rightarrow \mathbf{x}$. Then the reconstruction of attribute A of



(a) (b) (c)

Figure 3.1: (a): Point-based model of a toy leopard; (b) and (c): close view with increasing splat radius.

the surface at image point $\mathbf{x}$ is given by:

$$\mathcal{S}^A(\mathbf{x}) = \frac{\sum_i r'_i(\mathbf{x}) \mathcal{P'}_i^A(\mathbf{x})}{\sum_i r'_i(\mathbf{x})}. \tag{3.1}$$

with

$$r'_i(\mathbf{x}) = r_i(\mathcal{M}_i^{-1}(\mathbf{x})) \quad \text{et} \quad \mathcal{P'}_i^A(\mathbf{x}) = \mathcal{P}_i^A(\mathcal{M}_i^{-1}(\mathbf{x})). \tag{3.2}$$

In practice the support of the reconstruction kernels is truncated so that the above sum is finite. This resampling process of the surface attributes on the regular grid of image pixels is sure to produce aliasing artifacts, so Zwicker et al. [Zwicker 2001] convolve the reconstruction given by equation 3.1 with a screen space low-pass filter h. The final reconstruction is then given by:

$$\widetilde{\mathcal{S}}^A(\mathbf{x}) = (\mathcal{S}^A \otimes h)(\mathbf{x}) \approx \frac{\sum_i (r'_i \otimes h)(\mathbf{x}) \mathcal{P'}_i^A(\mathbf{x})}{\sum_i (r'_i \otimes h)(\mathbf{x})} = \frac{\sum_i \rho_i(\mathbf{x}) \mathcal{P'}_i^A(\mathbf{x})}{\sum_i \rho_i(\mathbf{x})}. \tag{3.3}$$

If the reconstruction kernels and the low-pass filter are both gaussians and if the projection is locally approximated by an affine transformation, the resampling kernels in image space are also gaussians. This yields elliptical image footprints that can be efficiently rasterized and blended. Several implementations of this rendering algorithm have been proposed, from the original CPU-based implementation of Zwicker et al. [Zwicker 2004] to hardware accelerated implementations making use of GPUs [Guennebaud 2004] [Botsch 2005]. Our own GPU implementation is very similar to [Botsch 2005] and consists in:

- projecting the model points and computing the gaussian kernels parameters,
- rasterizing the ellipse corresponding to the kernel footprint and blending the weighted attributes for each pixel of the ellipse,
- handling the visibility using back-face culling and ε-depth test techniques.

Figure 3.1 shows a leopard model rendered by our system.
In section 3.2.4, we extend this implementation to efficiently compute and interpolate dense 2D motion vector fields. First, the next section introduces our tracking framework.

3.2.3 Iterative Model-based Tracking with Keyframes

Let $\{I_t, t \in \mathbf{N}\}$ be a sequence of images featuring an object with 3D pose relative to the camera noted by $E_t = [R_t \mid t_t]$ at time t. The camera model is given by the 3×4 projection matrix $P_t = KE_t$, where K is the matrix of intrinsic parameters, which we assume constant over time. The goal is to recover the rigid transformation matrix E_t, whose unknowns are grouped into a vector $\beta_t \in \mathbf{R}^6$ corresponding three rotations and three translations parameters. We abusively call β_t the pose at time t.

The idea of tracking the object iteratively is to compute the pose estimation E_t knowing the previous pose E_{t-1} and a set of motion measurements made on the successive images I_{t-1} and I_t. We choose to track feature points, detected with the Harris criteria [Harris 1988] and matched using a standard correlation-based similarity measure which partially copes with illumination variation.

It is well known that such iterative update is prone to error accumulation and causes the recovered object pose to drift over time. As proposed by Vacchetti et al. [Vacchetti 2004] we overcome this problem by also registering the object with a keyframe. A keyframe is a view of the object where the pose is precisely computed offline. A set of keyframes is constructed in order to roughly cover the range of views that will be seen later in the tracked sequences. The registration with the keyframe is also performed by matching image features (in our case feature points), which now directly provide 2D-3D correspondences. One can view this a priori knowledge as introduced to limit the maximum drift allowed.

The pose update thus combines an iterative tracking part and a keyframe based part, which we describe more thoroughly in the two following subsections.

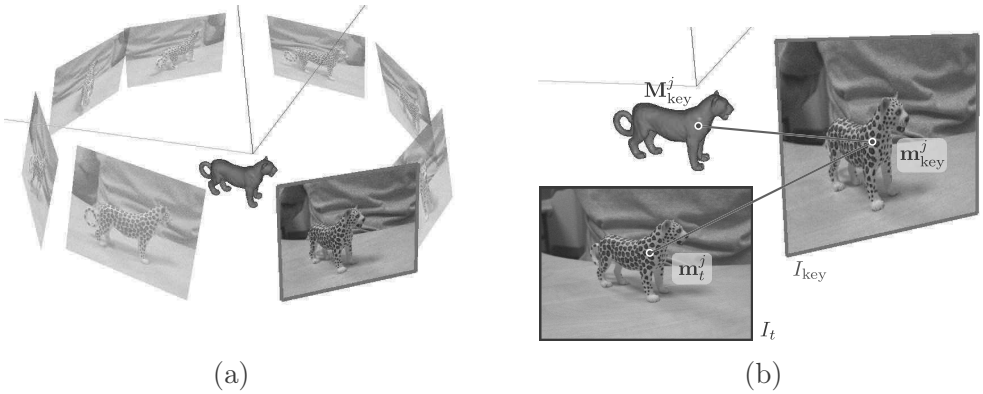
3.2.3.1 Iterative Tracking

Let $\mathbf{m}^i_{t-1}$ be a feature point detected on image I_{t-1}. If this point lies on the object image, then $\mathbf{m}^i_{t-1}$ is the image of a 3D point $\mathbf{M}^i$ on the object surface. Projecting $\mathbf{M}^i$ back on the current image I_t (with the unknown pose β_t) gives a point $\mathbf{m}^{i1}_t$. In essence any given pose β_t generates a 2D motion $\mathbf{d}_i = \mathbf{m}^i_t - \mathbf{m}^i_{t-1}$ in the image. The idea of iterative tracking is to relate this "predicted" motion with the measurement obtained by matching feature points. The problem stated above is a classic instance of bundle adjustment where both the pose β_t *and* the 3D points $\{\mathbf{M}_i\}$ have to be recovered. When a 3D model of the object is known the computation of the 3D points may be avoided. The whole process of projecting $\mathbf{m}^i_{t-1}$ on the model and then back on the image with a pose β can be modeled by a *transfer function* Ψ that relates $\mathbf{m}^i_{t-1}$ to $\mathbf{m}^i_t$ [Hartley 2003]. The pose β_t is then found by minimizing the distance between the ideal points and the matches:

$$\widehat{\beta_t} = \arg\min \beta \sum_{i=1}^{k} \|\Psi(\widehat{\beta_{t-1}}, \beta, \mathbf{m}^i_{t-1}) - \mathbf{m}^i_t\|^2. \tag{3.4}$$

In the case of a model defined by a polygonal mesh as used by Vacchetti et al., the 3D points lie on facet planes and $\Psi(\widehat{\beta_{t-1}}, \beta, \cdot)$ is a homography. In section 3.2.4 we will derive a transfer function suited to point-based models.

[1] For mathematical correctness we should note this point $\mathbf{m}^j_t$, with $j = \nu(i)$, however we can always reorder the point lists and keep the same superscript i for both corresponding points for simplicity.



(a) (b)

Figure 3.2: (a) A keyframe "cloud". (b): Registration with respect to a keyframe.

3.2.3.2 Keyframe based Tracking

The object pose is determined offline by hand or by using off-the-shelf registration techniques in each keyframes. A cloud of keyframes is represented in figure 3.2 (a), where a keyframe is represented in the object reference frame by a virtual camera. Feature points detected in a keyframe are back-projected onto the model and attached to the corresponding 3D points $\mathbf{M}^j_{\text{key}}$ (for this we use the ray-tracing method for point-based models described in [Adamson 2003]). Figure 3.2 (b) illustrates the registration process with respect to a keyframe I_{key}. Correspondences are drawn between feature points in the current frame and feature points in the keyframe. This gives 2D-3D correspondences that we use for registering the current pose by optimizing β_t for

$$\min_{\beta} \sum_{j} \|\Phi(\beta, \mathbf{M}^j_{key}) - \mathbf{m}^i_t\|^2 \tag{3.5}$$

where $\phi(\beta, \mathbf{M})$ is the projection of the point $\mathbf{M}$ according to the pose β. The above term can be added to the optimization function (3.4) or simply performed in a following step, as we do for reasons detailed in section 3.2.5.

There are two remaining issues. The first is to determine the closest keyframe to register. A simple and efficient way is to find the keyframe with the pose parameters β_{key} closest to the current pose β_{t-1} (according to the euclidean distance in $\mathbf{R}^6$). The second issue comes from the fact that the current frame and the closest keyframe may still be quite far apart, making feature points matching hard to achieve [Schmid 2000]. We address this by transforming the keyframe by rerendering the model in the pose β_t. Features are then matched between the current frame and this rerendered image (which acts like a proxy for the real keyframe).

3.2.4 Adaptation to a Point-based Model

Our first objective is to adapt the expression of the iterative update (eq. 3.4) to point based models. We derive a simple linear motion approximation taking the form of a basis of six 2D motion vectors. As opposed to the polygonal mesh setting, a motion model for image points can only be known at the projected 3D sample points. We thus rely on a dense reconstruction of the motion basis predictors to be able to query the predicted motion of any image point on the model projection.

3.2.4.1 Point based Motion Predictors

The rigid 3D motion between two successive images I_{t-1} and I_t is described by a three-dimensional euclidean transformation, composed of three elementary rotations and a translation, and modeled by a 4×4 matrix δE. Hence $E_t = \delta E E_{t-1}$.
To obtain a 2D motion model suited to the use of 3D points, we linearize the projection of the 3D motion as proposed by Drummond [Drummond 2002]. The idea is to express the projection of elementary 3D motions using the exponential map form of the euclidean matrix δE:

$$\delta E \approx I + \sum_{j=1}^{6} \alpha_j G_j \tag{3.6}$$

where the matrices G_i are the generators of 3D elementary motions (rotations and translations *w.r.t.* the axes of the object frame) and $\alpha = (\alpha_1, \cdots, \alpha_6)$ are the corresponding parameters of the translations and rotations.
We can now relate the apparent motion of a point $\mathbf{m}$ in the image to the small rigid 3D transformation of amplitude α_j undertaken by the object. By denoting $\mathbf{m} = P\mathbf{M} = (u, v, w)^\top$ and $PG_j\mathbf{M} = (u'_j, v'_j, w'_j)^\top$ we obtain the linear relation:

$$\mathbf{m}' = \mathbf{m} + \sum_{j=1}^{6} \alpha_j \mathbf{l}_j \text{ with } \mathbf{l}_j = \begin{bmatrix} \frac{u'_j w - u w'_j}{w^2} \\ \frac{v'_j w - v w'_j}{w^2} \end{bmatrix}, \ j = 1, \ldots, 6. \tag{3.7}$$

The 2D vector $\mathbf{l}_j$ is the first order approximation of the motion of the point $\mathbf{m}$, projection of $\mathbf{M}$ when $\mathbf{M}$ moves according to an elementary 3D transformation (either one of the three translations or rotations) of amplitude α_j in a frame attached to the object.

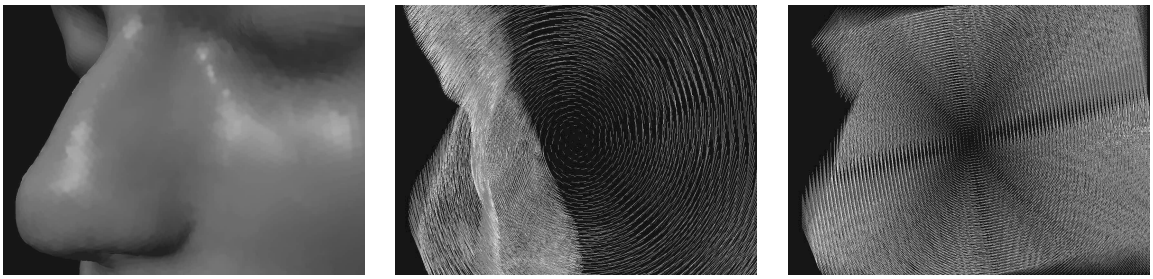
3.2.4.2 Dense Reconstruction of the Motion Predictors

Figure 3.3 illustrates the use of the splatting algorithm for computing a dense 2D motion field. Figure 3.3 (a) shows a rendered point-based 3D model of a face. For each sample point of the model, we can compute the vector $\mathbf{l}_6$ corresponding to the rotation around the z-axis (pointing towards the camera) (figure 3.3 (b)). Note that on this figure, motion vectors associated with hidden sampled points do also appear. In contrast, using the splatting algorithm, a dense motion field may be computed and depth is correctly handled (figure 3.3 (c)). For each pixel belonging to the projection of the object, $\mathbf{l}_6$ is computed using only visible neighboring splats.
Proceeding a step of iterative tracking is now possible using a linear approximation of the transfer function Ψ introduced in section 3.2.3.1. The motion vector approximation is defined on each meaningful pixel, that is, on a point of interest lying on the object projection. Let us denote $\{\mathbf{l}^i_j = (u^i_j, v^i_j)^\top, j = [|1, \ldots, 6\}$ the basis of elementary motions computed at the point $\mathbf{m}^i_{t-1}$, the pose is the solution of the problem:

$$\begin{cases} \min\limits_{(\alpha_1, \ldots, \alpha_6)^\top} \sum_{i=1}^{k} \| \sum_{j=1}^{6} \alpha_j \mathbf{l}^i_j + \mathbf{d}^i \|^2. \\ \mathbf{d}^i = \mathbf{m}^i_{t-1} - \mathbf{m}^i_t = (d^i_x, d^i_y)^\top \end{cases} \tag{3.8}$$

This problem is linear in the unknowns α_j, and may be written and solved in the following matrix form:

$$\widehat{\alpha_t} = \arg\min \alpha_t = (\alpha_1, \ldots, \alpha_6)^\top \| L\alpha_t + \mathbf{d} \|^2. \tag{3.9}$$



(a) (b) (c)

Figure 3.3: (a) A 3D point-based model of a face rendered with hidden surface removal. (b) Motion vectors $\{\mathbf{l}_6\}$ corresponding to a rotation of the model around the z-axis, some of which should be hidden. (c) The densely reconstructed motion field, showing vectors $\{\mathbf{l}_6\}$ interpolated by splatting. Note the correct handling of hidden points.

In practice a robust version of this linear estimate (via Iterative Reweigthed Least Squares) is used to cope with outlying matches. From this correction $\widehat{\alpha_t}$, we can easily derive an intermediate estimate $\widehat{\beta_t^1}$ of the unknown pose β_t. This estimate is then refined using keyframes.

3.2.4.3 Intermediate Keyframe Generation

As mentioned in section 3.2.3.2, the matches between $\{\mathbf{M}_i\}_i$ and $\{\mathbf{m}_t^i\}_i$ are obtained thanks to 2D-2D matches between the current frame I_t and a close keyframe. Finding matches between points of interest becomes difficult when the baseline between two views increases. The current frame and the selected keyframe are often in such a configuration. To adress this issue, an intermediate virtual keyframe is synthesized by splatting the model with the estimate $\widehat{\beta_t^1}$ of the current pose. Figure 3.4 illustrates the four steps of the whole process which we synthetise here:

1. Select the closest keyframe according to a similarity criterion between poses.
2. Extract texture information from the keyframe, thanks to the manually computed pose. Note that this step is done offline. Each keyframe thus provides its own textured version of the object model.
3. Render the textured model (texture chosen from the closest keyframe) according to the current pose estimate. This produces an intermediate keyframe containing only the tracked object (center image of figure 3.4).
4. Determine 2D-2D matches between the current frame and the intermediate keyframe and deduce 2D-3D correspondences for the current frame.

We have two ways to refine the initial guess $\widehat{\beta_t^1}$. First, a direct minimization (eq. 3.4) can be performed with a few iterations of Levenberg-Marquardt. A second option is to use again the linearized approach described in section 3.2.4.3. Thanks to the estimated $\widehat{\beta_t^1}$, we can not only render the textured model of the object but also reconstruct the motion predictors (see central figure 3.4) *for this resynthesized view*. With the 2D-2D matches between I_t and this resynthetized view (see dashed blue line on figure 3.4), we now have the information to formulate a problem similar to eq. (3.9), giving a correction to the current pose estimate $\widehat{\beta_t^1}$. This refinement process

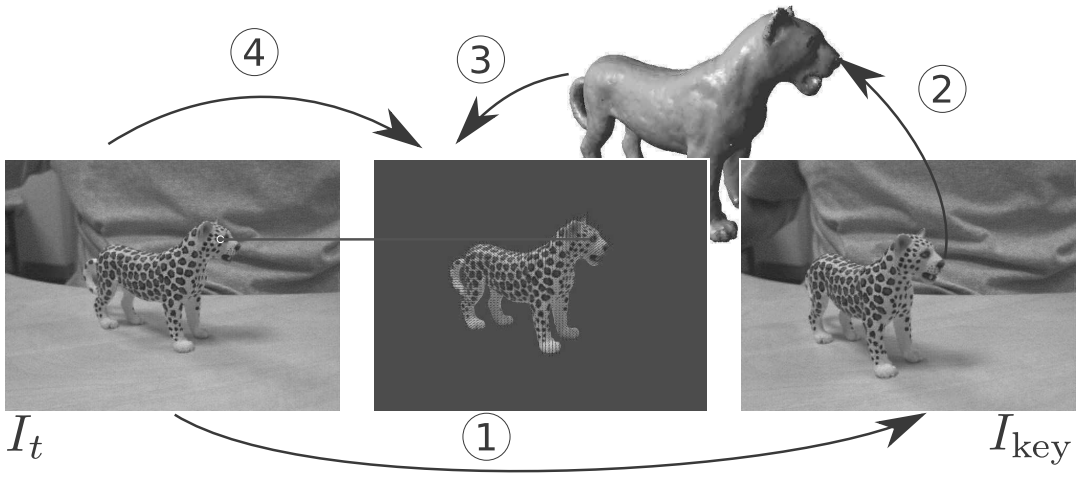


Figure 3.4: Illustration of the use of keyframes. See text for details.

may be iterated (as suggested by Drummond [Drummond 2002]) but we found out that a single step suffices, as shown the experiments described in next section.

3.2.5 Implementation Details and Experiments

We designed a complete implementation of the tracking framework presented in the previous sections, consisting of the following steps. Assuming the pose β_0 of the object in frame I_0 is known, we then detect feature points in each frame I_t and match them to the points from the previous image I_{t-1}. We use a standard correlation-based technique on small 9×9 windows with a cross-validation step. These two steps are implemented on the GPU using GPGPU techniques. We then render the dense motion vectors as maps for the pose $\widehat{\beta_{t-1}}$ according to section 3.2.4.2 and download them to the CPU memory. This allows us to sample the maps at feature points located on image I_{t-1}, giving the predicted motions used to form the matrix L in equation (3.9). By solving problem (3.9), we obtain an intermediate pose estimate $\widehat{\beta_t^1}$, which we use as described in section 3.2.4.3 (keyframe resynthesis and pose refinement using the linearized approach).
The following graph (figure 3.5) sums up the timings of the different parts of the algorithm, measured on a Core2 2.6GHz system with a G80 graphic card. The overall pose update process is on average 81.5ms, which allows real-time interaction with the system.
We now present experiments highlighting the benefits of using a hybrid tracking approach. Iterative tracking, while robust in presence of incorrect image measurements, is prone to accumulate estimation error and the reprojection of the tracked object will often lag behind its real counterpart. In contrast keyframe based tracking does not accumulate error over time but due to the less precise matching process with the synthesized view, it is affected by jittering and can also fail when not enough correct matches are found. We illustrate these observations on a synthetic sequence which provides ground-truth (GT) values for the pose parameters. The sequence involves translations and rotations (see figure 3.6 (a)). We plot the output of the algorithm against ground-truth values when activating iterative tracking only (IT), when activating keyframe tracking only (KF) and finally when combining both (KF+IT). The graphs (b)-(e) of the recovered pose parameters show that the combined approach is always closer to the ground truth. The results on figure 3.6 (d), corresponding to the rotation around the y axis are particularly significant: the iterative tracker looses track early and never recovers; the keyframe-based

Operations	time (ms)	
(1) GPU color splatting	2	
(2) GPU vector reconstruction	17	×2
(3) GPU point detection	1.5	
(4) GPU point matching	15	×2
(5) Iterative pose estimation	7	×2
(6) Pose refinement w/ keyframe	7	
Total	81.5	(12 fps)

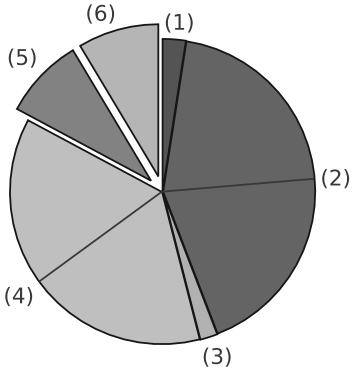


Figure 3.5: The timings of the different parts of our tracking algorithm. The operations done on the CPU (in shades of green) have been offset from the GPU based ones (in shades of blue).

tracker exhibits a lot of jitter; on the contrary the combined tracker output is at the same time smooth and close to the ground-truth.
We finally show images from our tracker running in real-time on live sequences taken by a standard camera. Some screen captures are presented on figure 3.7. Figure 3.8 shows a direct application of the tracking to augmented reality.

3.2.6 Conclusion, limitations and perspectives

We have proposed an original extension of an existing method and a real time implementation for 3D tracking of rigid objects with a single camera. We drew our work upon a sound mathematical framework of surface splatting. This technique, well known in computer graphics, allows us to derive an algorithm which combines iterative tracking and keyframes, achieving similar precision compared to previous methods. Finally, we showed real-time performance on complex freeform objects.

The perspectives of this work are twofold. First we would like to investigate the formulation of an edge-based tracking technique on point-based models. This would further prove the interest of such models in this context. Second, we believe that this work is an interesting path to address 3D non-rigid tracking. Deformations in 3D could indeed correspond to 2D deformation fields of the same nature as those we generate by splatting rigid motions. Working with a point-based 3D model will simplify the management of visibility including self-occlusions. The density of such 2D motion fields allows the combination of texture and geometry for non-rigid tracking, and can be used to verify the optical flow constraint equation or more general illumination constraints.

In the next section we also start from a predefined 3D shape, but instead of considering motion analysis we perform shape analysis.

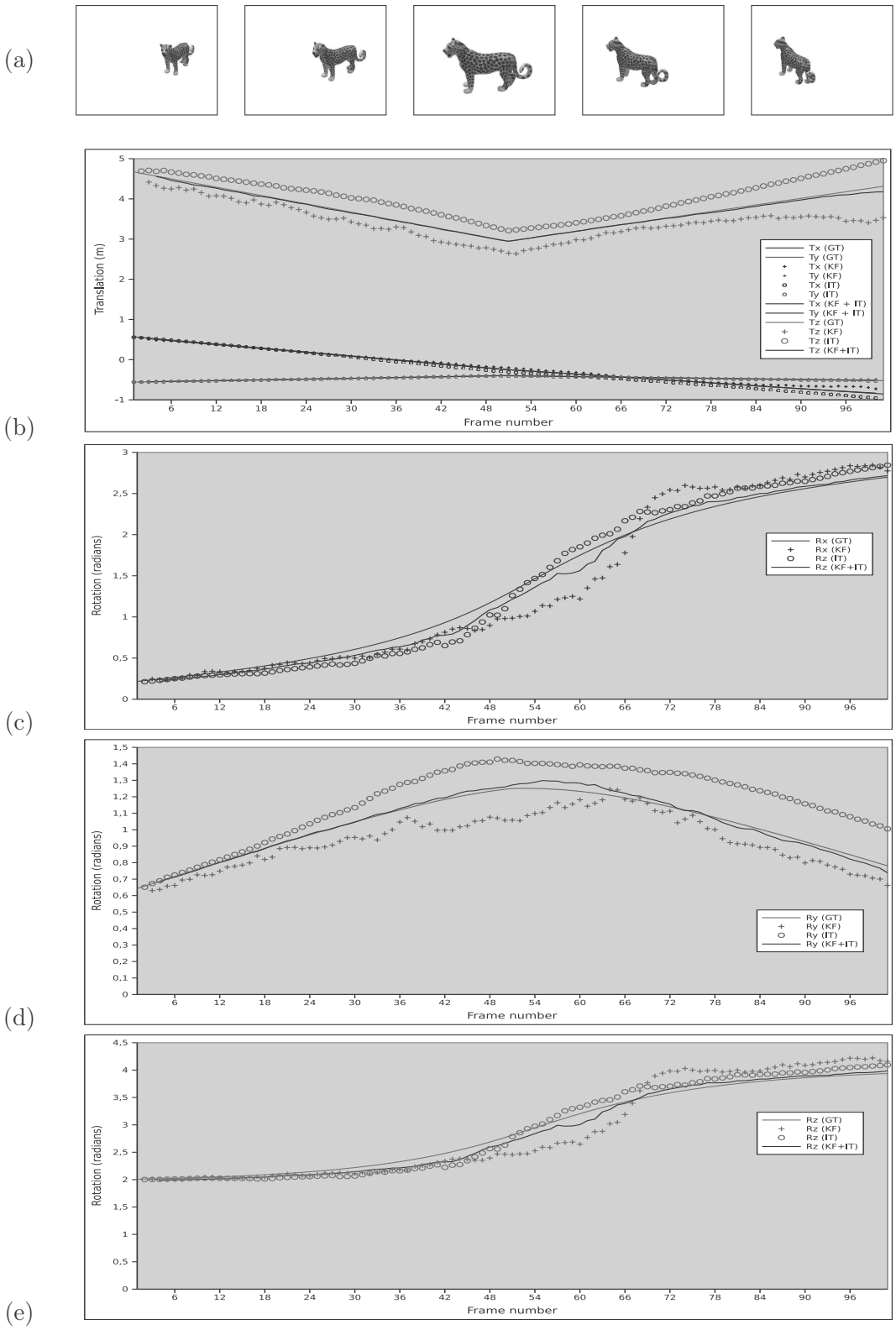
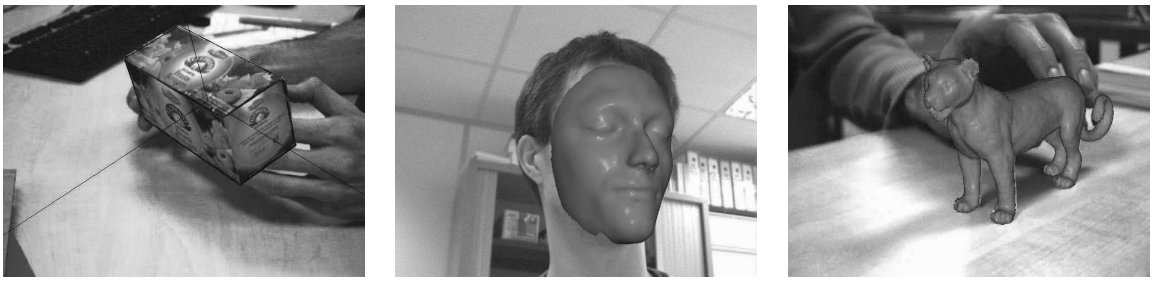


Figure 3.6: Tracking results for the synthetic sequence shown in (a). The graph (b) shows the evolution of all 3 translation parameters. Graphs (c), (d) and (e) show the individual rotation parameters (respectively Rx, Ry and Rz).



(a) (b) (c)

Figure 3.7: Screenshots from live sequences, with three models of increasing complexity.

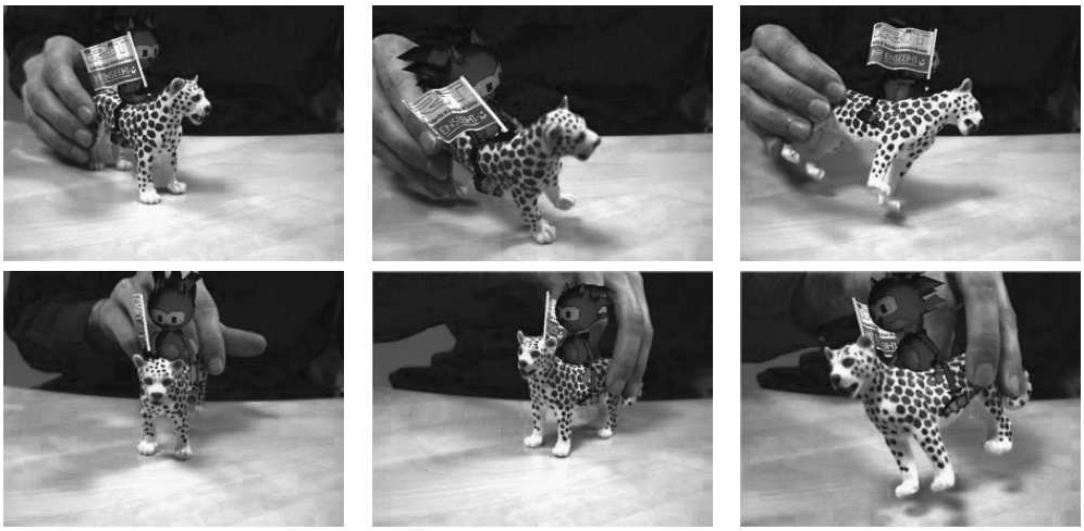


Figure 3.8: Application of the tracking to a Augmented Reality application; the 3D nature of the tracked model permits taking into account the occlusion of virtual objects (the rider) by real object.

3.3 Similarity detection in parametric surfaces

Our contributions are two fold: we adapt the current technique called votes transformation space for parametric surfaces and we improve the identification of isometries.

3.3.1 Context and motivation

Analysis of similarity within a 3D shape is a first step for many applications in the manipulation of 3D shapes. First, for editing: considering a coherent edition of similar parts within a model is natural. For example, when editing the head of a 3D character it is natural to apply the changes following the symmetry of the face. When editing one of the ears, the same changes should be applied to the other one, respecting the initial symmetry. The similarity analysis is also relevant for the creation, or design of 3D models: [Iyer 2005] claims that 75% of design activity involves reusing existing shapes (or starting from existing models) to design a new shape. Another application is model compression. For providing a compact representation, identifying similar parts can certainly help compress a model; when two parts are similar up to a transformation, only one of them can be coded, and coding the transformation is then enough for generating the second part. This compact representation may then be helpful for transmitting the model at lower cost (we will talk again about this perspective in section 4.3.7).

Our work proposes a framework for detecting local similarities in free-form parametric models, in particular on B-Spline or NURBS based B-reps: patches similar up to an approximated isometry are identified. Parametric surfaces, in particular Non-Uniform Rational B-Spline (NURBS), provide a powerful tool in the hands of the academic and industrial communities concerned with the design and analysis of objects [Dimas 1999]. NURBS based B-reps (Boundary representations) are industrial standards and are widely used in different domains such as molecular chemistry [Bajaj 1997], 3D geographical information systems [Caumon 2003] and mechanical components design [Chu 2006]. With the particular attractiveness of NURBS surfaces in 3D design industry, the similarity detection would certainly be useful. Besides, parametric NURBS representations allow to easily and reliably access differential information over the surface. Their representation by control points also gives the designer intuitive control.

3.3.2 State of the art

In recent years, many articles have been published on similarity detection both in 2D image processing and in 3D modeling. In a first approach, Zabrodsky et al. [Zabrodsky 1995] quantified existing symmetries within 2D and 3D objects, using a metric called the *symmetry distance*. The symmetry distance of a shape is defined to be the minimum mean squared distance required to move points of the original shape in order to obtain a symmetrical shape. Sun et al. [Sun 1997] converted the symmetry detection problem into the correlation of Gaussian images; rotational and bilateral symmetries are identified by applying orientation histograms.

For 3D shape matching, two dominant techniques were proposed. First, global feature-based techniques represents 3D objects as a set of global features, for example, reduced feature vectors [Cardone 2006]. The other set of methods uses graph-based techniques: the solid models are converted into attributed graphs that represent the geometrical and topological relationship between models entities [Hilaga 2001, Ma 2010]. However, in both cases, these techniques

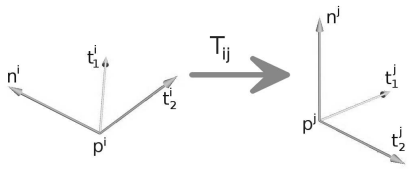


Figure 3.9: Local Frames of two similar points p_i et p_j according to right hand rule.

can neither identify similar parts within a model nor compute the transformation between these similar parts. Recently, many papers proposed to identify similarities within 3D meshes [Kazhdan 2004, Podolak 2006, Berner 2008, Bokeloh 2009, Lipman 2009, Mitra 2013] with different approaches like planar-reflective symmetry, graph-based matching, or votes transformation space. Kazhdan et al. [Kazhdan 2004] introduced a *reflective symmetry descriptor* that represents a measure of reflective symmetry for an arbitrary 3D model for all planes through the model's center of mass. Podolak et al. [Podolak 2006] generalized this approach to identify symmetries of 3D objects associated with an arbitrary plane. Graph-based approach requires detecting local features on 3D shape from which a neighborhood graph is build to describe the coarse scale similarity structure of the object. Berner et al. [Berner 2008] perform subgraph matching in graphs of feature points while Bokeloh et al. [Bokeloh 2009] apply feature lines.

Some authors [Lipman 2009, Mitra 2013] have applied a new technique for symmetry detection that we call *votes in transformation space*. This technique bears some similarity to the Hough transform: points on the model with similar features are paired. A pair of points is associated the transformation between the two points and attached local frames; these transformations are cast to a transformation space and where they form a constellation of transformation votes. Clusters of these votes are candidates for defining similar parts in the model. While Mitra et al. [Mitra 2013] use Euclidean transformations as the feature to extract similarity, Lipman et al. [Lipman 2009] adopt Möbius transformations.
The votes transformation space attracts our interest since it allows to retrieve a large class of potential transformations and it is able to identify similar parts in existing 3D objects and to characterize the transformation. In order to give a general view of this scheme, we detail the algorithm proposed in [Mitra 2013] that consists in the following four steps:

1. *Sampling and analysis:* a set of points is sampled over the surface of a 3D object. Since point positions are not sufficient to determine a general Euclidean transformation, geometry features at each sample are computed (the principal curvatures and a local frame composed of the principal directions and a normal vector). The signature is the couple of principal curvatures; points on the surface are paired if they have the same signature.
2. *Pairing:* each pair of points is associated a transformation corresponding to a vote in transformation space. Given two points p_i and p_j with their local (orthonormal) frames consisting in two tangents and a normal (figure 3.9), the transformation T_{ij} is computed so that p_i and its frame are mapped into p_j and p_j's frame. This transformation is then cast into votes of transformation space Γ.
3. *Clustering:* in transformation space Γ, each point T_{ij} represents a transformation between two similar points. Hence, clusters of similar transformations are identified since they may characterize two similar parts of the object.

4. *Validation:* ideally, a cluster of the previous step is a set of point pairs which belong to a couple of surface patches similar up to a transformation close to the cluster. However, spatial coherence between point pairs is lost in transformation space. Thus, this step enforces spatial coherence of the point pairs by applying an incremental region growing algorithm.

Proposed Pipeline for Isometry Detection

Our work aims at identifying surface patches in a B-rep model that are similar up to an approximated isometry (we do not consider scaling). To identify the similarities, we adapt the *votes in transformation space* that are used successfully for 3D meshes [Lipman 2009, Mitra 2013]. The adapted pipeline consists in five consecutive steps (see Figure 3.10); the following subsections correspond to the five steps of the pipeline.

3.3.3 Computation of the Signatures

In our setting, we work with B-rep models based on trimmed free-form patches made of NURBS tensor product surfaces. For the first three steps of the similarity detection pipeline, it is sufficient to consider the patches independently. Thus, in this section, we focus on NURBS tensor product surfaces and in particular in computing a set of sample points and their characterizing signatures.

3.3.3.1 NURBS based models

Let S be a tensor product NURBS surface of bi-degree (p, q) associated to two knots vectors $\mathbf{u} = \{u_0, \ldots, u_n\}$ and $\mathbf{v} = \{v_0, \ldots, v_m\}$ and a set of control points $C = \{P_{ij} \mid i \in [0, n-p], j \in [0, m-q]\}$ weighted by $w_{ij} \in \mathbb{R}$, defined by the following equation:

$$S(u,v) = \frac{\sum_{i=0}^{n-p} \sum_{j=0}^{m-q} N_{i,p}(u) N_{j,q}(v) w_{ij} P_{ij}}{\sum_{i=0}^{n-p} \sum_{j=0}^{m-q} N_{i,p}(u) N_{j,q}(v) w_{ij}}. \tag{3.10}$$

In a B-rep model, faces are not only represented by this type of NURBS, but also by other types such as planes, cylinders or spheres. However, one of the advantages of NURBS is that we can represent free-form as well as quadric surfaces (e.g. [Farin 2002a]).

3.3.3.2 Local differential properties: computation of the signature

Any point on the parametric surface, corresponding to a parametric coordinates (u, v), is attached to a set of persistent properties which is called *the signature* at that point. In our work, the signature at each point is composed of the two principal curvatures and an *orthonormal affine frame* having origin at that point, the unit vectors are the normal vector and the two principal directions (i.e. tangent vectors associated to the considered principal curvatures). The signature computation at a specific point on NURBS surface is based on local differential properties that could be evaluated from *the first and the second fundamental form* [Struik 1961, Farin 2002a]. The *first fundamental form* that describes completely the metric properties of a surface, is defined as the distance of two points on a curve of the surface:

$$ds^2 = E\ du^2 \ + \ 2F\ du\ dv \ + \ G\ dv^2 \tag{3.11}$$

where $E = S_u \cdot S_u$, $F = S_u \cdot S_v$, $G = S_v \cdot S_v$, and ds is also called *the element of arc.*

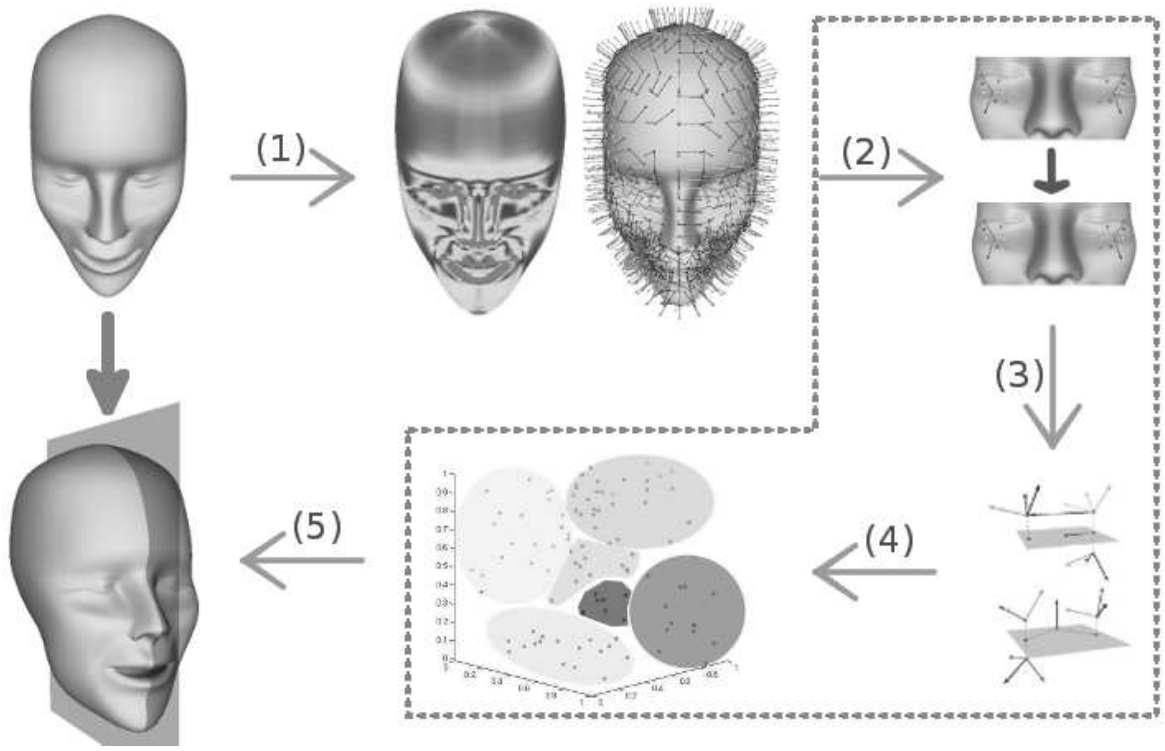


Figure 3.10: Proposed pipeline: (1) points are sampled over all B-reps of a CAD model by a sampling technique that adapts the parametrization; (2) when the signature at each point is computed, vector directions are determined by parametrization, so it is not a geometric property of the surface. For this reason, local frames are not coherent, in particular to identify indirect isometries. We propose then a simple method to overcome this problem; (3) Instead of working in a single transformation space, isometries between pairs of points are partitioned into five types, based on orientation and on their fixed points. The isometries classification has two advantages: first it simplifies the clustering (next step), but it also maps the pairs in transformation spaces of reduced dimensions. In this pipeline, the computation of the transformations is a major concern that affects considerably the quality of the result. By parameterizing the isometries differently, we improve the identification of isometries; (4) spectral clustering is applied in these five different spaces to extract the evidence of existing similarity in the model. Unlike Mean Shift algorithm, our approach is fully unsupervised, and as such, is able to group automatically clusters without customizing global parameters; (5) similarities among local patches are identified following an adaptive growing process adapted for multiple faces in B-rep models.

The first fundamental form states that, for a given point p, partial derivatives S_u and S_v generate a tangent plane to the surface of origin p. Hence, the unitary normal vector is:

$$n = \frac{S_u \wedge S_v}{\left\| S_u \wedge S_v \right\|} = \frac{1}{\sqrt{EG - F^2}} \left(S_u \wedge S_v \right) \tag{3.12}$$

It associates to non normalized vectors S_u, S_v to form an affine frame of origin p.
Next, *the second fundamental form* of a parametric surface is defined by:

$$\kappa \cos\phi ds^2 = Ldu^2 + 2Mdudv + Ndv^2 \tag{3.13}$$

where $L = S_{uu} \cdot n$, $M = S_{uv} \cdot n$, $N = S_{vv} \cdot n$, and S_{uu}, S_{uv}, S_{vv} are second partial derivatives at p.
Equation (3.13) means that, for a given direction du/dv in u, v plane and a given angle ϕ, the *first and second fundamental forms* allow us to compute the curvature κ of a curve lying on the

surface.
For this reason, two symmetric matrices are introduced:

$$\mathcal{F}_1 = \begin{pmatrix} E & F \\ F & G \end{pmatrix} \text{ and } \mathcal{F}_2 = \begin{pmatrix} L & M \\ M & N \end{pmatrix} \tag{3.14}$$

Because S_u and S_v are linearly independent, $\mathcal{F}_1$ is always invertible. The matrix $\mathcal{F}_1^{-1}\mathcal{F}_2$ is also symmetric and so always has real eigenvalues and orthogonal eigenvectors. The two eigenvalues κ_1, κ_2 are the two *principal curvatures* and the two eigenvectors $t_1 = (\xi_1, \eta_1)^T$, $t_2 = (\xi_2, \eta_2)^T$ define the two *principal directions*:

$$\begin{aligned} t_1 = &\quad \xi_1 S_u + \eta_1 S_v \\ t_2 = &\quad \xi_2 S_u + \eta_2 S_v \end{aligned} \tag{3.15}$$

For points such that ($\kappa_1 = \kappa_2$), that is *umbilical points*, principal directions are not uniquely defined, thus we do not consider them. For other points, the orientation of t_1 and t_2 depends on the parametrization.
Every point on the surface that is associated to a signature characterized by its local differential properties, might be potentially sampled for later computations. By benefiting from the facilities offered by parametric surfaces, a net of sample points on the surface is obtained by sampling the parametric domain. The sampling affects the following steps of the algorithm in two ways. First, the denser sampling is, the better result is. Second, the denser samples also worsen the performance. For this reason, we evaluate a net of points uniformly on the surface but select randomly a limited number of samples following a uniform law on this point net.

3.3.3.3 Robust surface orientation

Two sample points p_i and p_j are considered similar if their principal curvature matches, that is, $\kappa_1^i \sim \kappa_1^j$ and $\kappa_2^i \sim \kappa_2^j$. Two similar points are paired to evaluate the transformation between them. However, the orientation of the vectors t_1 and t_2 depends on the parametrization. So, a coherent orientation of the frame is necessary, in particular to distinguish direct from indirect transformation. We choose an orientation of the normal vector coherent for the whole surface, but we modify the direction of tangent frame vectors. For each pair (p_i, p_j), we identify the orientation of principal vectors at p_j that is the most coherent to direction associated to those at p_i. Suppose that the frame at point p_i is fixed, in other words, the direction of vectors t_1^i and t_2^i is arbitrarily fixed. Consider now the frame at p_j. We can observe that there are four possible different orientations of principal vectors (tangents) at p_j. We project the neighbors

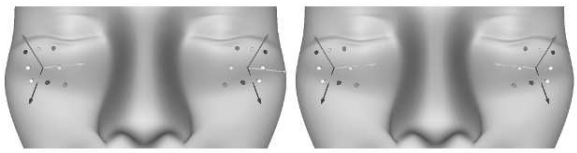


Figure 3.11: On the left: the orientation of the frame vectors follows the parameterization, so the two frames are not symmetric. On the right: we propose to find a coherent orientation of the vector frames by analyzing the points neighbors. Now, the two frames are symmetric, as is the underlying surface.

of p_i into the tangent plane, and order them into a sequence by turning around p_i. This gives us a reference list of curvatures. The four lists of neighbors of p_j corresponding to the four possible orientations of t_1^j and t_2^j are compared to the reference list. The chosen directions are thus the ones that minimize the sum of squares of differences between its list and the reference

list. Figure 3.11 shows a case of a plane symmetry where the initial orientation of vectors would have led to identifying a (wrong) direct transformation between points p_i and p_j.
The proposed orientation technique, independent of the parametrization, allows to identify direct versus indirect transformations. We will further refine the classification of the isometries in the next section.

3.3.4 Isometry Spaces

Instead of considering all transformations in a 6-dimensional transformation space as in [Mitra 2013], we first partition the isometries and map them into one of the five isometry spaces. The advantage of these classifications is two fold: it simplifies the clustering, but also, it expresses the transformation in a space of lower dimension. As an example, clustering translations in the original 6-dimensional transformation space requires the clustering algorithm to discriminate between points that belong to a degenerated 3-dimensional subspace. In our approach, the clustering will be applied directly in this subspace, taking into account only the relevant parameters.

3.3.4.1 Computation of the isometry

Given a points pair (p_i, p_j) as in the figure 3.9, we would like to evaluate the transformation from p_i to p_j so that p_i move to p_j's position and that the computed orthonormal frame at p_i aligns to the frame at p_j. We denote R_{ij} the rotation between these two frames and t_{ij} the corresponding translation. The computation is as follow:

$$R_{ij} = \begin{bmatrix} n^i \\ t_1^i \\ t_2^i \end{bmatrix}^T * \begin{bmatrix} n^i \cdot n^j & n^i \cdot t_1^j & n^i \cdot t_2^j \\ t_1^i \cdot n^j & t_1^i \cdot t_1^j & t_1^i \cdot t_2^j \\ t_2^i \cdot n_j & t_2^i \cdot t_1^j & t_2^i \cdot t_2^j \end{bmatrix} * \begin{bmatrix} n^i \\ t_1^i \\ t_2^i \end{bmatrix} \tag{3.16}$$

$$t_{ij} = p_j - R_{ij} * p_i \tag{3.17}$$

The transformation R_{ij} is an orthogonal matrix, i.e. $R_{ij} \in O(3)$, thus $T_{ij} : p_i(n^i, t_1^i, t_2^i) \longmapsto p_j(n^j, t_1^j, t_2^j)$ is then an isometry. Hence, T_{ij} belongs to $Is(X)$, the isometry group. We denote $\overrightarrow{T_{ij}}$ the associated linear transform, that is, the transform of matrix R_{ij}.

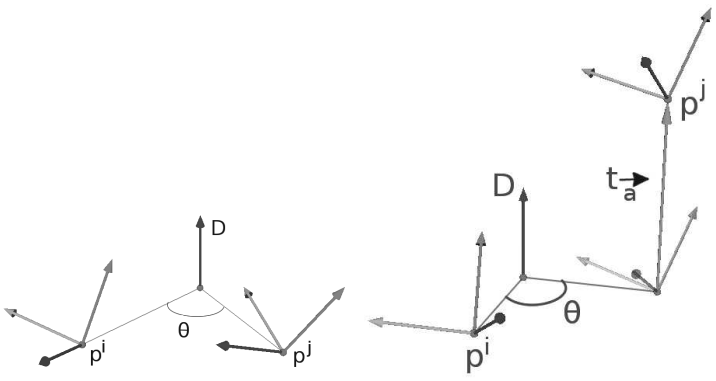
3.3.4.2 Classification of isometries

The classification of the affine isometries in 3D is based on the following classical theorem (see e.g. [Tisseron 1988]).

Theorem
Given $T \in Is(X)$, there exists a unique couple $(g, t_{\overrightarrow{a}})$ where g is an isometry having a non empty set of fixed points G and here $t_{\overrightarrow{a}}$ is a translation of vector $\overrightarrow{a} \in \overrightarrow{G}$, the vector space associated with G, such that $T = t_{\overrightarrow{a}} \circ g$. Additionally:

- $T = g \circ t_{\overrightarrow{a}}$ and $\overrightarrow{G} = E(1, \overrightarrow{T})$, the vector subspace associated with the eigenvalue 1.
- $T = g$ and $\overrightarrow{a} = 0$ if and only if T has at least one fixed point.
- If T has no fixed point, $dim G \geq 1$.

Let $\overrightarrow{T}$ the linear transformation associated with the affine isometry T. The transformation T is direct if $det(\overrightarrow{T}) = 1$ and T is indirect if $det(\overrightarrow{T}) = -1$. Let $\overrightarrow{G} = E(1, \overrightarrow{T})$ and $\alpha = dim \overrightarrow{G}$. If



(a) Line of fixed points (b) No fixed point

Figure 3.12: Classification of Direct Isometries based on fixed points. Local frames consist of a normal (red vector) and two principal directions (blue and green vectors).

$\alpha = 1$ or 2, let , and $\overrightarrow{a}$ the projection of $\overrightarrow{MT(M)}$ on $\overrightarrow{G}$ for an arbitrary M. We can deduce the isometry type of T depending on its fixed points (or on α and $\overrightarrow{a}$), as follows:

Direct Isometry

1. *A line* (D) *of fixed points* ($\alpha = 1$, $\overrightarrow{a} = 0$): T is a rotation around the line (D) directed by $\overrightarrow{n} \in E(1, \overrightarrow{T})$ (figure 3.12a).
2. *No fixed point* ($\overrightarrow{a} \neq 0$): T is either a translation of $\overrightarrow{a}$ ($\alpha = 3$) or the composition of a rotation around (D) directed by $\overrightarrow{a}$ and a non-zero translation colinear to (D) ($\alpha = 1$) (figure 3.12b).

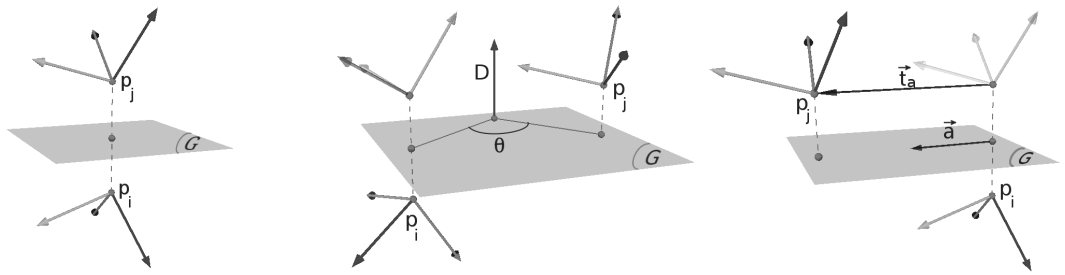
Indirect Isometry

1. *A plane* G *of fixed points* ($\alpha = 2$, $\overrightarrow{a} = 0$): T is a reflexion relative to the plane G of direction $\overrightarrow{n}_{1,2} \in E(1, \overrightarrow{T})$ and orthogonal to $\overrightarrow{n} \in E(-1, \overrightarrow{T})$ (figure 3.13a).
2. *A unique fixed point* A : ($\alpha = 0$) T consists of a rotation around an axis (D) directed by $\overrightarrow{n} \in E(-1, \overrightarrow{T})$ and passing through A, and a reflexion relative to a plane G containing A and perpendicular to (D) (figure 3.13b).
3. *No fixed point* ($\alpha = 2$, $\overrightarrow{a} \neq 0$): T is composed of a symmetry relative to a plane G whose normal $\overrightarrow{n} \in E(-1, \overrightarrow{T})$, and a non-zero translation $t_{\overrightarrow{a}}$ parallel to this plane (figure 3.13c).

We now can classify the isometries into five groups depending on their orientation and fixed points.

3.3.4.3 Robustness of the isometry classification to pair of points orientation

We test the robustness of the classification of isometries to the reorientation, and show that we only need to orient consistently around 80% of the pairs of points to get a correct identification of the isometries.
We proposed three B-rep models of leaves as showed in the figure 3.15. Given N_{Exp} the number of *expected re-oriented* pairs and N_{Total} the *total number of pairs* computed in each model. Then, the rate R is the ratio between these two factors.



(a) Plane of fixed points (b) A unique fixed point (c) No fixed point

Figure 3.13: Classification of Indirect Isometries based on fixed points. Local frames consist of a normal (red vector) and two principal directions (blue and green vectors).

Model	N_{Exp}	N_{Total}	R
3.15a	520	621	0.84
3.15b	509	618	0.82
3.15c	505	621	0.81

Table 3.1: Ratio of well oriented points using the *surface orientation* algorithm.

According to table 3.1, our test cases shows that this algorithm has a tolerance rate up to 80%. Despite the orientation still failed at points whose the opposite neighbors (symmetric via these points in the parameters net) are similar, the validation step guarantees that the *classification of isometries* is robust.

3.3.4.4 Comparison of two isometries

We now have five different transformation spaces, and for each, will apply a clustering algorithm. The clustering algorithm works on a metric space, so we define a distance on each of these spaces, that is, we derive a distance between two isometries of the same type.

For direct isometries, the components of isometries are the rotation axis (D) and angle θ, and the translation $t_{\overrightarrow{a}}$. Note that the rotation axis and the translation have the same direction, so it is sufficient to compare the angle between the lines. For comparing the rotations we use this angle and the distance between the two axes; for translations, we still compare the length of the translation vectors (the angle is the same as for the axes).

For indirect isometries, the analysis is identical to the direct setting, except for the symmetry plane G. The comparison between planes consists in comparing the normals to these planes and computing the distance between the mid-point and the plane.

In the following, we denote $d(T, T')$ the distance between the two isometries T and T' corresponding to the two point pairs (p_i, p_j) and $(p_{i'}, p_{j'})$; M, M' the midpoints of $[p_i, p_j]$ and $[p_{i'}, p_{j'}]$; $dist(P, G)$ denotes the distance from a point, line or a plane to another one. We keep the same notation as in the previous section for T and take (D'), G', $\overrightarrow{n'}$, $\overrightarrow{a'}$, θ' for T'.

Direct isometries

$$\begin{aligned} d(T,T') = \quad & (1 - |cos(D,D'|) + \tfrac{|(\theta-\theta')|}{2\pi} \\ & +\omega_1 dist(D,D') + \omega_2 |(\|\overrightarrow{a}\| - \|\overrightarrow{a'}\|)| \end{aligned} \tag{3.18}$$

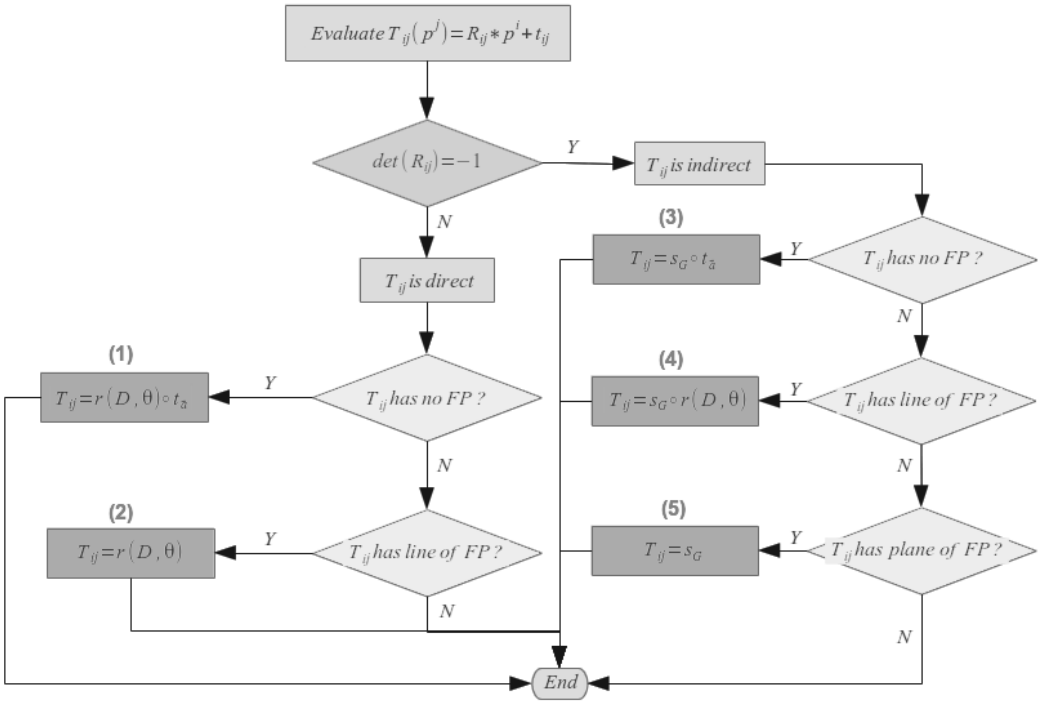


Figure 3.14: The algorithm to classify the isometries into one of the five types, s_G is a symmetry relative to the plane G; $t_{\overrightarrow{a}}$ a translation in the direction of vector $\overrightarrow{a}$; $r(D,\theta)$ a rotation axis around axis (D) of angle θ.

Indirect isometries

$$\begin{aligned} d(T,T') = \quad & (1-|cos(\overrightarrow{n}\cdot\overrightarrow{n'})|) \\ & +\omega_1(dist(M,G')+dist(M',G)) \\ & +\frac{|(\theta-\theta')|}{\pi} \\ & \omega_2|(\|\overrightarrow{a}\|-\|\overrightarrow{a'}\|)| \end{aligned} \tag{3.19}$$

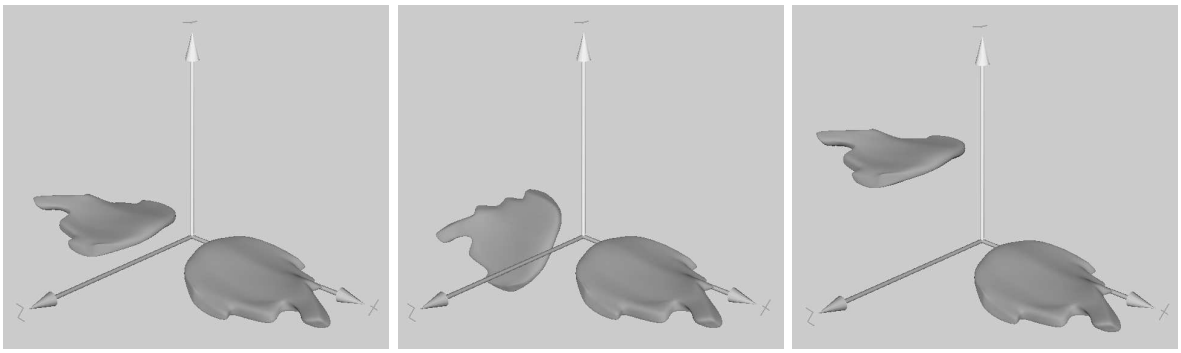
The weight ω_i are chosen as the inverse of the diagonal of the bounding box of the model.

3.3.5 Clustering

After computing the isometries as described in the previous section (Section 3.3.4), the clustering step aims at grouping pairs of points having approximatively the same isometry. This step is based on a spectral approach called spectral clustering and differs from the Mean Shift algorithm [Mitra 2013] which requires difficult parameters tuning. The advantage of our approach is to be unsupervised, so no parameters tuning is necessary, and in particular, the number of cluster does not need to be known in advance. The number of cluster is determined by the heuristic proposed in [Mouysset 2011]. Figure 3.16 compares a cluster of a transformation (here a planar reflection) identified with Mean Shift or spectral clustering for a same number of clusters, and using the original, six dimensionnal transformation space.

3.3.6 Validation

Ideally, every class obtained by the clustering is a set of point pairs which belong to a couple of surface patches similar up to an approximated isometry. However, spatial coherence between point pairs is lost during the isometry clustering. Therefore, the purpose of the *validation* is



(a) Pure reflexion. (b) reflection + rotation. (c) reflection + translation

Figure 3.15: Proposed CAD models representing the three possible types of indirect isometries for the *surface orientation* algorithm test cases.

to overcome this problem in order to identify similar patches. We present the validation step within a NURBS patch (section 3.3.6.1) and then consider region growing over a B-Rep model, which may include multiple NURBS patches (section 3.3.6.2).

3.3.6.1 Validation within a NURBS patch

The validation is performed by a region expanding process. Given C_k, a class of points pairs in an isometry space, a pair (p_i, p_j) is selected randomly. The chosen isometry T_{ij} is applied to the eight neighbors of p_i, their images are thus compared to eight neighbors of p_j. If the deviation of any neighbor is under a chosen threshold, the points pair is accepted as belonging to the two similar patches. This process continues iteratively; we further test the neighbors of p_i. It is repeated until all points on the surface are visited, or the condition does not hold, or until all pairs in class are considered. This step generates a candidate for two similar patches. Nevertheless, this process stops at the boundary of the surface. But a 3D object modeled by NURBS based B-rep is composed by several NURBS surfaces.

3.3.6.2 Validation within a B-rep

The figure 3.17 represents an overview of the B-rep specification in the context of OpenNURBS. In fact, a NURBS based B-rep is a set of trimmed NURBS that consists in a surface and some trimming contours. The trimming contours define which parts of the surface are kept or removed. In OpenNURBS context, the *loop* is an abstraction of a trimming contour. It is defined by a set of closed trimming curves that are in turn expressed by *trims*. Each *trim* is attached to a 2D curve and an *edge*. The 2D curve defines the curvilinear coordinates of the *trim* within the surface. The *edge* is a 3D curve on the surface and is a boundary. Furthermore, an edge can be shared among multiple trims. Given p the point on the boundary of the surface where the validation cannot continue. The proposed algorithm 1 identifies a point q on an adjacent surface close to p.

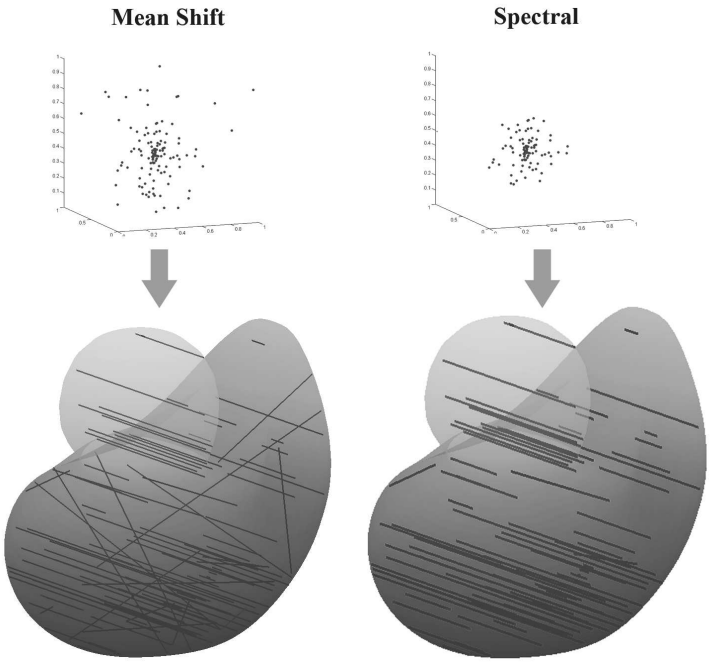


Figure 3.16: On the left: a cluster of transformations identified using Mean Shift, and the corresponding pairs of points. On the right: The corresponding cluster identified with spectral clustering, and its corrsponding pairs of points. We see that some noisy transformation has been removed from the cluster.

Algorithm 1 Identification of adjacent point

```
Find the closest edge e to p
if e is shared with other face then
    Determine the adjacent face S
    Take the set P of points on all edges of S
    Find q the closed point to p in P
    Find curvilinear parameters of q
```

3.3.7 Experiments

We have implemented the pipeline described in section 3.3.2 to identify the similar patches within the following B-Rep models. We use CAD models under OpenNURBS specifications[2] for our experiments. In the following, we propose some test scenarios to validate these tools following by the results on some CAD models of our pipeline.

Next, by applying our algorithm of *Automatic Spectral clustering* [Mouysset 2011], the results of clustering in the figure 3.18 illustrate the effectiveness between *Euclidean transformation* approach [Mitra 2013] and our approach of *classification of Isometry*. This figure represents three leaves in a model that have two symmetric pairs of leaves. Besides, every line that connects two points having the same signature corresponds to a point in transformation space. Additionally, lines with the same color are in the same cluster (i.e. the same transformation in general). In this test case, we use the computation of Euclidean transformation in a six dimensionnal space and the distance metric as proposed by Mitra et al. [Mitra 2013]. We can observe that while

[2] `http://www.opennurbs.org/`

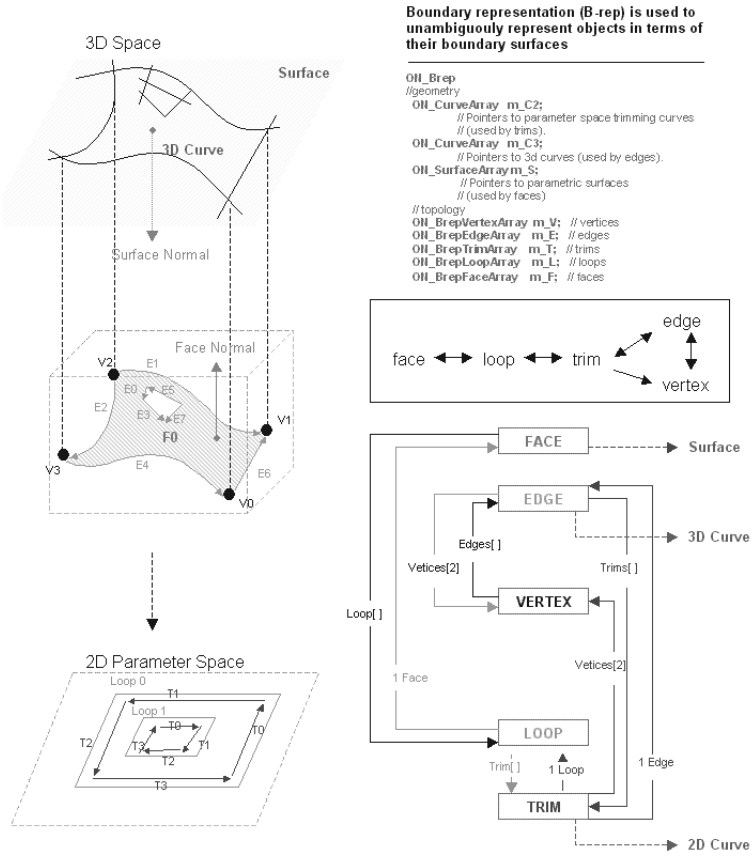
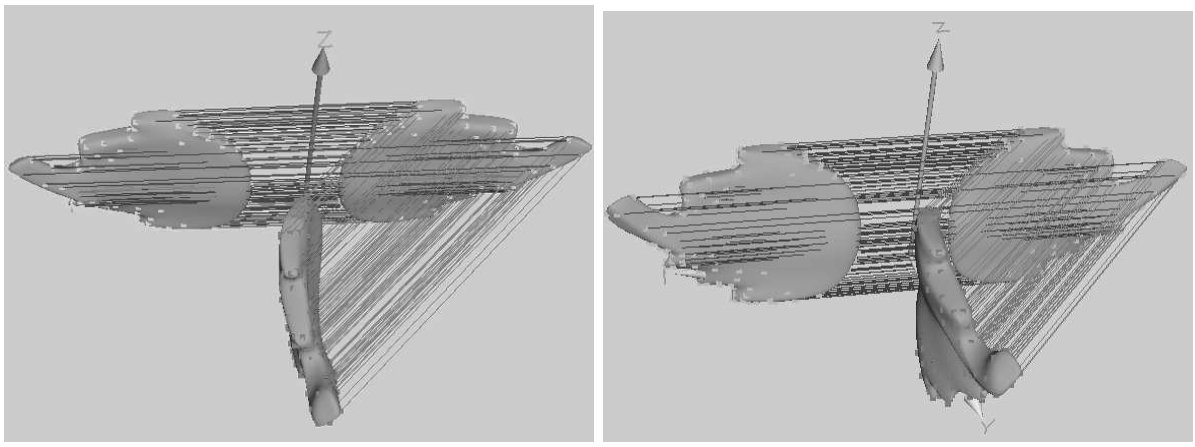


Figure 3.17: Boundary representation (B-rep) in the context of OpenNURBS (from http://wiki.mcneel.com/).

there are some wrong classified points in the *Euclidean transformation* approach (figure 3.18a), our approach can address this problem (3.18b). In other words, with the aid of the classification of isometries, the output of the clustering algorithm was significantly improved. Moreover, the use of *Automatic Spectral clustering* algorithm also contributes to the robustness of our pipeline. In fact, the results shown in this figure are obtained without any parameter tuning.

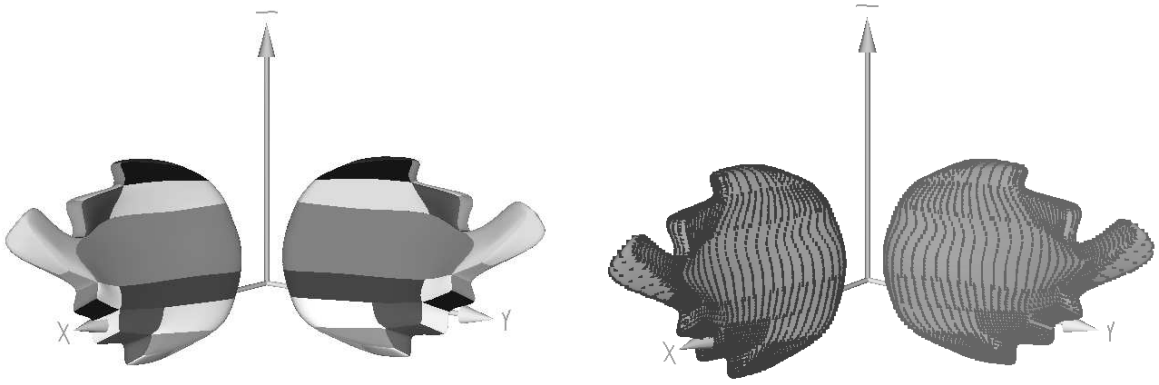
Figure 3.19 presents a result of the validation process within B-rep. The figure 3.19a shows that there are two separated B-reps that are formed by several trimmed NURBS surfaces displayed by different colors. As in the figure 3.19b, the validation has successfully validated all the points over the surface of these B-rep objects.

Finally, the figures 3.20, 3.21, 3.22 and 3.23 represent the final results of our experiments on some CAD models downloaded from GrabCAD (http://grabcad.com/). These results represent different isometries detected by our proposed pipeline. The first set of leave models exhibit the indirect isometries. In fact, while the figure 3.20a shows a symmetry, the figure 3.20b represents a symmetry following by a rotation axis, and a symmetry following by a translation is detected in the figure 3.20c. Also, the figures 3.22a and 3.22b describe the direct isometries between the four legs of a dragon: this isometry is decomposed into a translation and a rotation. The figures of the plane and the sunglasses demonstrate the symmetry between different parts in these models. In addition, the figure 3.21a also demonstrates a direct isometry composed by a rotation axis between the two parts of the plane tail. Next, Figures 3.23a and 3.23b describe some similarity detection results given a model representing a series of human head in a model,



(a) (b)

Figure 3.18: Comparison of the effectiveness between the *Euclidean transformation* approach 3.18a and the *classification of Isometry* approach 3.18b.



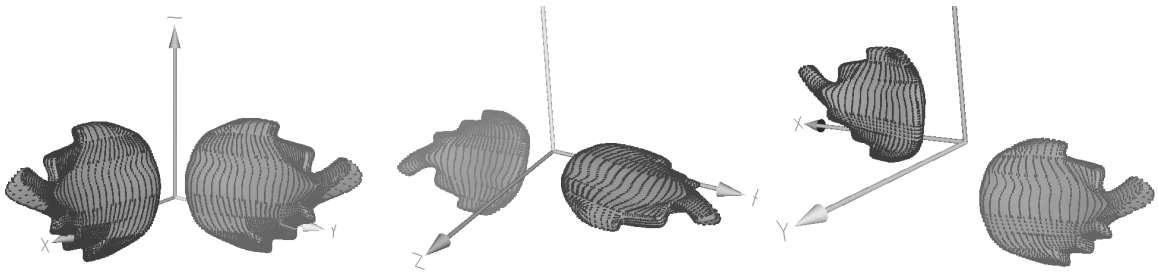
(a) Original B-reps (b) Result

Figure 3.19: Result of validation within a B-rep.

in which, from left to right, every head presents a refinement step on the surface. In other words, there are some minor deformations between these heads. When applying our pipeline, one of the identified transformations is the translation between the green dots and the blue dots (figure 3.23a), another is the symmetry inside a B-rep (figure 3.23b). This result demonstrates that our pipeline also detects approximate isometries between similar, but not exactly equal, surfaces.

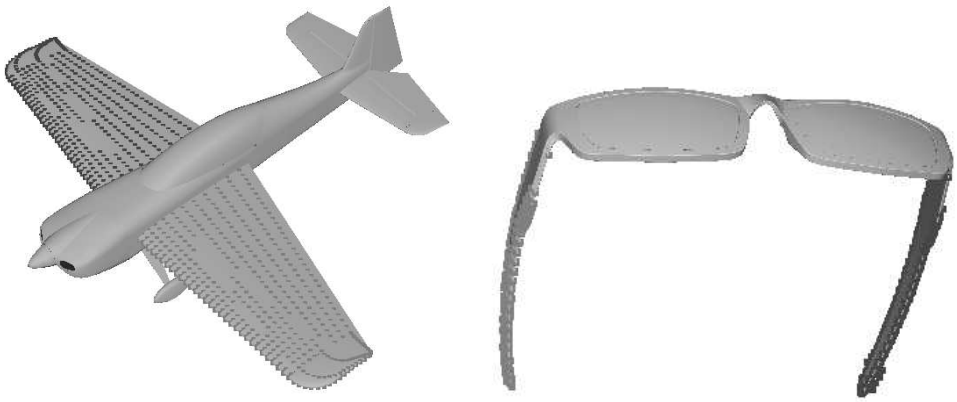
3.3.8 Conclusion, limitations and perspectives

In this article, we propose an algorithm to identify similar parts within objects modeled by NURBS based B-Reps, by adapting and improving the *votes transformation space* approach described by Mitra et al. [Mitra 2013]. Beside adapting the approach for parametric representations, we have proposed a local coherent frames orientation simply based on the points neighbors. A (robust) globally coherent orientation is then insured at the validation step. The local orientation allows to sort *direct* and *indirect isometries*. Furthermore, based on the analysis of fixed points, local isometries are further partitioned into five subsets. The experiments show that this classification before the clustering steps significantly improves the results. Further-



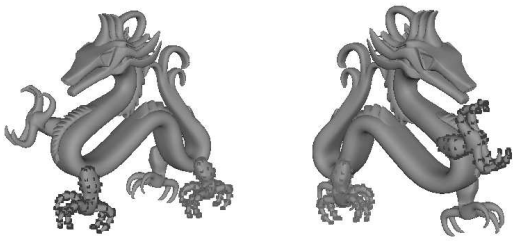
(a) (b) (c)

Figure 3.20: Similarities in leaves models.



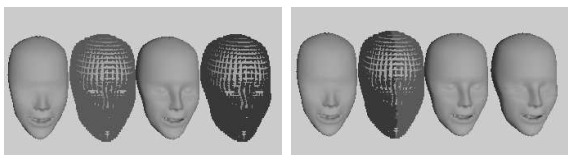
(a) (b)

Figure 3.21: Symmetry detected in models.



(a) (b)

Figure 3.22: Direct isometry detected in models.



(a) (b)

Figure 3.23: Similarities detected in a model of human heads.

more, the clustering was further improved by using a fully unsupervised spectral method, for which, unlike for the *Mean-shift* algorithm, parameter tuning is not necessary. In particular, the number of isometries (clusters) to be identified does not need to be known in advance. Finally, the validation step extends the identified isometries among different NURBS patch within the B-rep. We are now able to recover isometric patches of B-splines or NURBS surfaces or similar to an isometry, or an approximate isometry (like shown in the experiment section). For future work, first we would like to filter the similarity detection by filtering similarities between control points. Second, we would like to exploit the isometries for applications: by linking the control structures corresponding to these patches, to offer the possibility to coherently edit or segment the models. Moreover, we could use the similarity to limit the storage size of the model.

3.4 Easing interactions with 3D models using crowdsourcing

3.4.1 Crowdsourcing

Crowdsourcing is an emerging research paradigm for motivating many common users to contribute their knowledge and expertise in completing tasks, traditionally performed by a designated agent (e.g. [Wang 2010]). From its definition, crowdsourcing has been used as a technique in which competence and expertise distributed among the members of the crowd are aggregated and exploited, e.g. the case of Wikipedia.
In [Zhao 2012], Zhao et al proposed a conceptual framework for crowdsourcing. One common way to apply crowdsourcing theory in the field of multimedia is to provide a set of tools and mechanisms, allowing many common users to interact/play with multimedia content, and the analysis of these interactions allows to enhance the multimedia content. As an example, in [Carlier 2010], we analyzed users interactions in the context of zoomable video: a HD video was displayed to users in a tool allowing zoom and pan. The analysis of their interactions leads to the identification of relevant sequences and regions of interest. More generally, crowdsourcing is expected to provide a resource for relevant multimedia adaptation, traditionally created by experts of the field. Social networks as Wikipedia and Facebook are examples of crowdsourcing platforms. In both cases, users are volunteers to create and share their knowledge and opinions using supported tools and services. Our method belongs to this crowdsourcing for multimedia setting, targeting 3D content. We show that if 3D object has particularly important regions, user navigation can be used to ease subsequent Web3D user experience, by helping them accessing relevant parts in less time, and with simplified interaction.

3.4.2 A proof of concept: getting knowledge from user interactions with 3D models

We proposed a first step towards easing Web3D user navigation using crowdsourcing. We collect user interactions to determine informative regions (the most popular regions) within a 3D object. We show that subsequent users can benefit from previous user navigation to improve their Web3D experience: a set of 3D poses is computed and used for recommendations to subsequent users. The recommended poses aim at simplifying the 3D navigation, that is, reducing the number of pan, zoom and rotate events the user may need to reach his/her desired regions. An experimental user study shows that our system redirects user to interesting areas quicker, with fewer interactions required.
We present how crowdsourcing can be used as a simple method to capture meaningful parts, called ROIs (regions of interest). A set of 3D poses, called *Recommended Views* are computed based on previous users navigation, and then suggested to subsequent users.

3.4.2.1 Interacting with Online 3D content

Some early work for navigation [Chittaro 2002, Hughes 2002] exploited the adaptation capabilities of VRML models. Chittaro proposed some guidance for navigation in a 3D environment, and Hugues proposed to include annotation within the 3D world. Whereas the adaptation of the interaction is relevant to our work, we target common place applications (such as e-commerce) where users are interacting with a singe 3D object and are not necessarily familiar with 3D navigation. For this reason we also chose to implement a Web3D application.
In [Kim 2004], Kim et al. introduced multi-resolution models for the selection of adaptive 3D presentation taking into account network constraints and terminal characteristics. This tech-

nique requires a trade-off between the realism of 3D objects and the rendering/transmission performance. [Laga 2010] applied semantic-driven approach to analyze the 3D shapes and extract their meaningful features for the best view selection. The accuracy of this method, however, depends on the nature of 3D shapes and the computation process is often long and heavy. [Chittaro 2007] points out that the specificity of 3D interactions requires to develop a specific adaptation framework for 3D content. Our approach proposes such an adaptation: by analyzing the interactions of the first users with 3D content, we propose an adapted presentation and navigation to the subsequent users.

3.4.2.2 Analyzing user interactions to simplify the navigation

Approach Overview We propose a new paradigm to improve Web3D user experience. First, we show that user navigation traces can be used to detect interesting regions of the 3D model. For that, we artificially create a ROI on a 3D model by adding stamps (see figure 3.24). We then show that by analyzing the navigation of a limited set of users, we can automatically give recommendations to the subsequent users to ease their navigation: for them, we propose a framework with recommendations, and quantify by a user study the performance of users given recommendations.

Figure 3.24: 3D Models with added stamps.

Computing recommended views takes three steps (see figure 3.25). First, we collect 3D interactions from a set of initial users (first column). Collected data, including visible 3D points, camera position and normal vectors on each user created frame, are sent back to server and stored into database. This is our crowdsourcing part. The second step analyzes the logged traces to automatically detect ROIs, presenting the parts on the surface of the 3D object which appears the most through crowd navigation (second column). The goal is to identify informative regions containing high density of logged 3D points. Second column of figure 3.25 shows two examples of detected ROIs marked by two red circles: each contains our stamp on the chest and on the knee of the statue. In the last step, we compute corresponding poses, *Recommended views* such that the ROI is at the center of the viewport (third column). Recommended Views are then proposed to the next generations of users via a set of clickable images so that they may use or not.

System Architecture Our system acts as an extension for the existing Web3D application (figure 3.26) using x3dom as a rendering engine[3]. On the client side, the *Listener* component

[3] `http://www.x3dom.org`

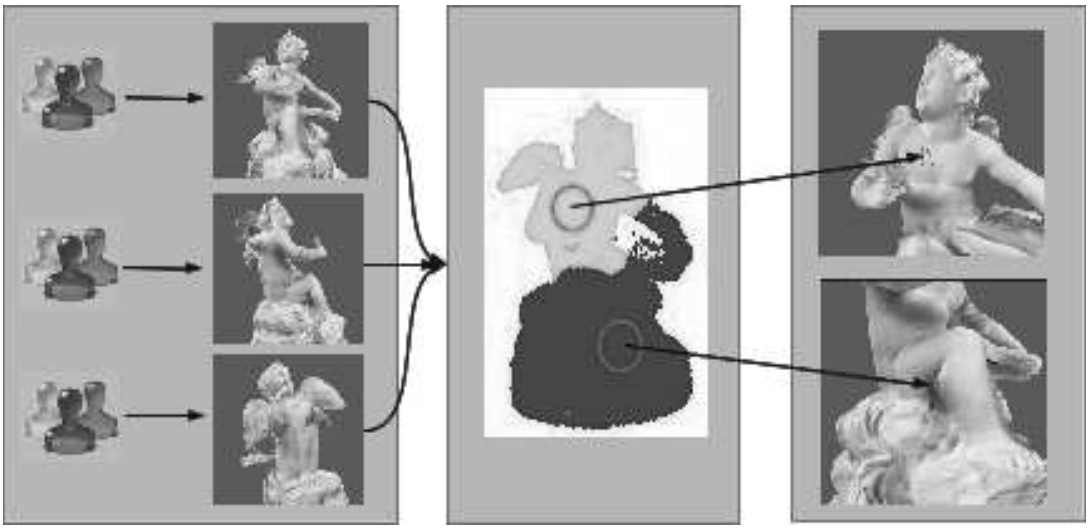


Figure 3.25: Overview: the recommended view are computed in three steps, from left to right.

captures user interactions and sends data back to the server by remote requests (XmlHttpRequest). On the server side, the *Communication Module* acts as a interface between the Web3D application, client requests and our system core (ROI Detector, Generator, Rating Module). In the system core, the *ROI Detector* analyzes the logged data to determine informative regions which are then used by *Generator* to compute recommendations. The other component in the system core is the *Rating Module* which collects crowd feedback and opinions to continuously enhance recommendations.

Analysis of User Traces

We collect visible 3D points on each frame using the 3D picking buffer technique proposed by [Behr 2010]. The scene is rendered into a texture attached to a framebuffer object: at each frame, the normalized world coordinates of 3D points corresponding to visible 2D points on the canvas are encoded into the RGB channels. At each frame, we also extract the camera position from the current view matrix and compute the distances between the camera and the visible points. A shader program captures the normal vector at each visible 3D point.

In order to characterize the ROIs from the collected 3D points, we use a Mean-shift clustering technique (e.g. [Comaniciu 2002]). We chose Mean-shift for its robustness (we do expect outliers in users navigation). Two points need to be addressed when using Mean-shift: (1) the metric of the feature space, and (2) the shape of the kernel. For (1), fortunately, our feature space is Euclidean since logged points are on the surface of real 3D objects. We choose a flat kernel for (2) to ensure the convergence of Mean-shift. The output of the algorithm is a set of clusters, presenting detected ROIs. Each cluster contains a mode and the corresponding data points.

The area around the mode is the most informative region of the cluster. The mode's position is intrinsically very close to the object surface thanks to the metric and the flat kernel properties. Each cluster gives a ROI defined by its center, the point of the collected data nearest to the mode, and its size, the number of points in the cluster.

Recommended View Generation

We compute recommendations such that the ROI is at the center of the viewport. The method is to set camera position and orientation so that it looks towards the ROI center. To do so, we first extract the logged normal vector at the ROI center and set it as the camera look-at vector. Next,

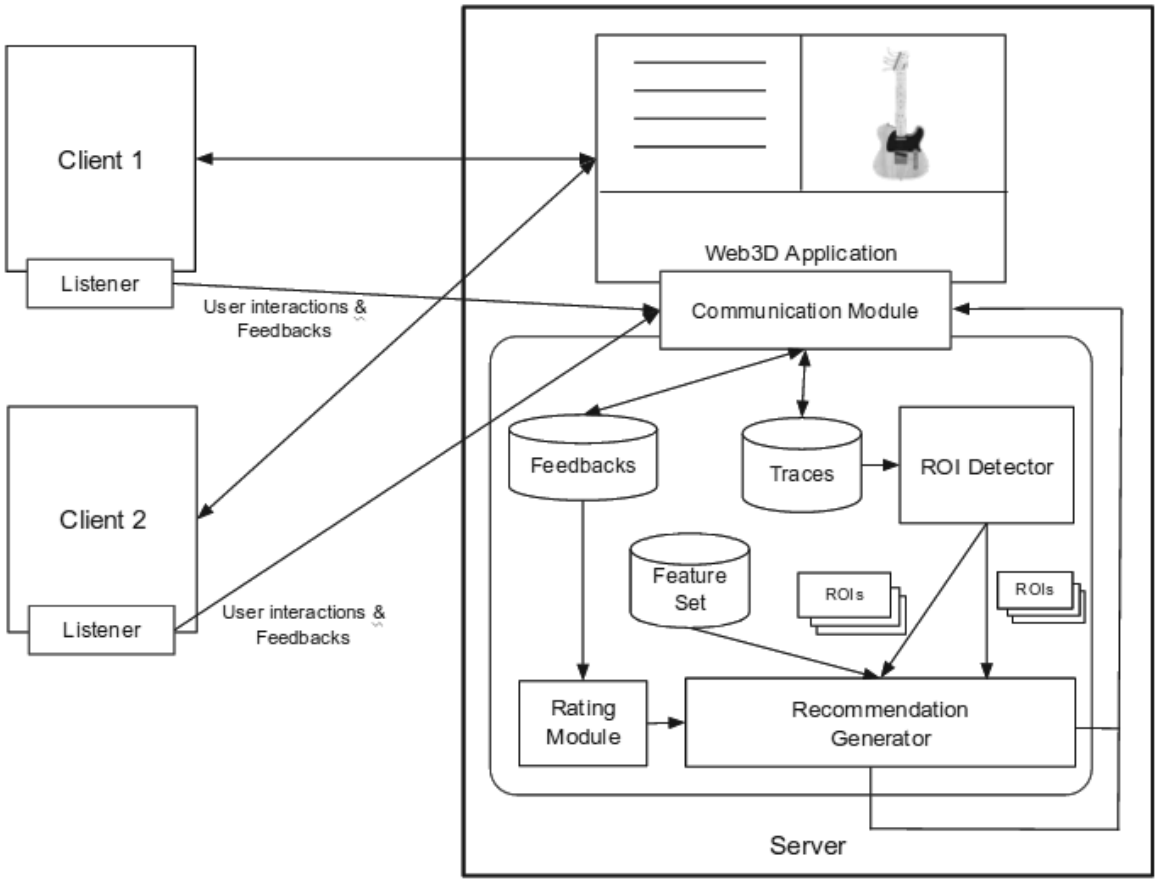


Figure 3.26: System Architecture.

we get the median of the collected distances between the ROI center and the camera. Using this median, the ROI center and the normal vector, we calculate the camera position. Each time there is a request for recommendation from users, a new camera with computed orientation and position is set to generate the recommended 3D view.
We introduce recommendations to subsequent users with a set of clickable buttons to swap between recommended viewpoints (as shown in Figure 3.27). Interactive navigation is, however, still possible.

3.4.2.3 Experiments

In this section we describe the experimental setup of the user study and an analysis of the results in order to evaluate our method.
Set-up of the Experiments In this project, the user study has two main goals. We need to collect interactions of the first sample of users in order to generate recommended 3D views: this is the crowdsourcing part. Then in the second step, we propose recommendations to another sample of users that will allow us to properly evaluate the effectiveness and the quality of these recommendations. This user study was performed on a total number of 28 users, 18 of them being part of the crowdsourcing step and 10 others evaluating the recommendations. 17 users were male, 11 were female and their age ranges from 23 to 35.

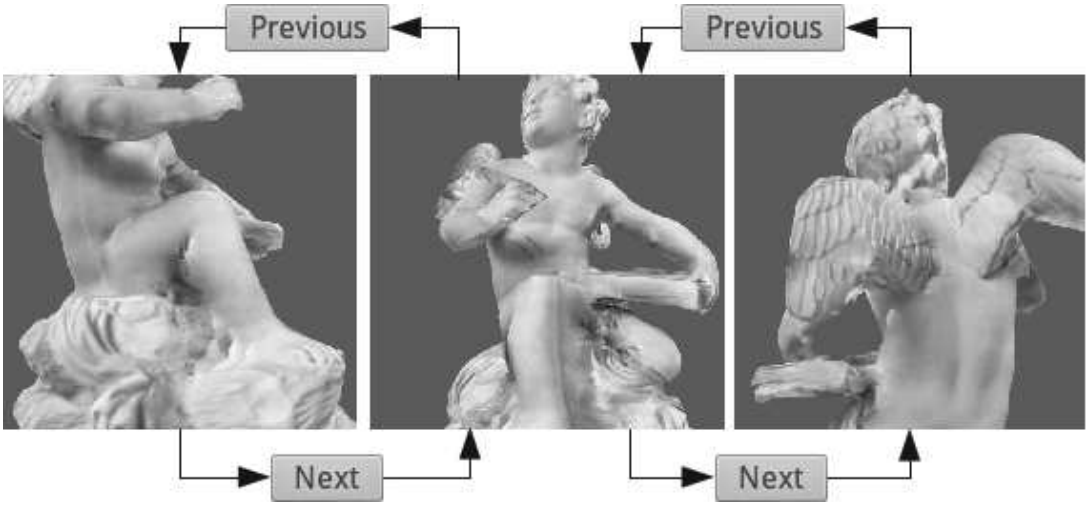


Figure 3.27: Recommended Views accessed via Clickable Buttons.

The interface allows users to zoom, pan and rotate around 3D objects. We recorded every interaction from users into our database. We used two 3D models from *3DCOFORM* (see figure 3.24) for the user study, called *ManStatue* and *VaseWhite*. *ManStatue* is a rather complex model whereas *VaseWhite* is a simple model, presenting a lot of symmetry.

The purpose of this user study is simple. We provided the crowd an explicit task so as to motivate them into their navigation. We gave them 6 models of ancient statues, called *ManStatue*s (MS1, MS2 and MS3) and *VaseWhite*s (VW1, VW2 and VW3) respectively. Each statue has a stamp containing information about its manufacture date at different places; on the knee in *MS1*, on the chest in *MS2*, on the left wing in *MS3*, on the white part in *VW1*, on the yellow part in *VW2* and on the top in *VW3*. A statue is considered as more valuable if its manufacture date is older. The motivation task provided to the crowd is to locate the stamps in order to get the oldest manufacture date presenting the most valuable statue.

The first step of our user study was divided into three successive parts. First we explained our interface to users and let them try it by themselves on an additional 3D model. Then we introduced the motivation task as well as showed them what the stamp looks like on a 3D model. At this point we want users to fully understand what is expected from them before executing the task. Finally we provided the users with the six test models, in a random order to avoid some undesirable side effects. We recorded the time and interaction taken by users to complete every task.

In the second step of the user study, we modified the interface to introduce the recommendations (Figure 3.27). Apart from that, we followed the exact same protocol for the second step of the user study as for the first step.

Results The recommendations where helpful to simplify the recovery the locations of the stamps. It shows that characterizing ROIs from user navigation traces is possible. Table 3.2 shows the average time in seconds taken by users to locate the stamps on the 3D models at each step of the user study. First, we see that looking for the stamps on a single model takes in average almost a minute of interactions. This is an important fact, showing that freely interacting with an online 3D model is indeed cumbersome. Second, clearly, the users from the step 2, who were given recommendations based on traces of users from the step 1, are quicker to perform the task than users from the step 1. This result presents that our recommendations are able to guide

users to the regions of interest of the 3D models. Moreover, table 3.3 shows that users generally used less interactions to reach the target during the step 2 of the user study than during the step 1. The recommendations may also help users in reducing negative interactions in case of heavy 3D data.

	MS1	*MS2*	*MS3*	*VW1*	*VW2*	*VW3*
Step 1	76	48	67	53	50	54
Step 2	11	9	8	7	8	6

Table 3.2: Average time (in seconds) taken by users to complete tasks, for each step of the user study.

	MS1	*MS2*	*MS3*	*VW1*	*VW2*	*VW3*
Step 1	192	108	153	128	126	133
Step 2	29	22	20	17	21	15

Table 3.3: Average number of zoom, pan and rotate events created by users to complete tasks, for each step of the user study.

The results show that we were able to generate useful recommendations by gathering data from 18 users. It is necessary to know if our recommendations would converge to the same regions of interest even if we were to use less users.

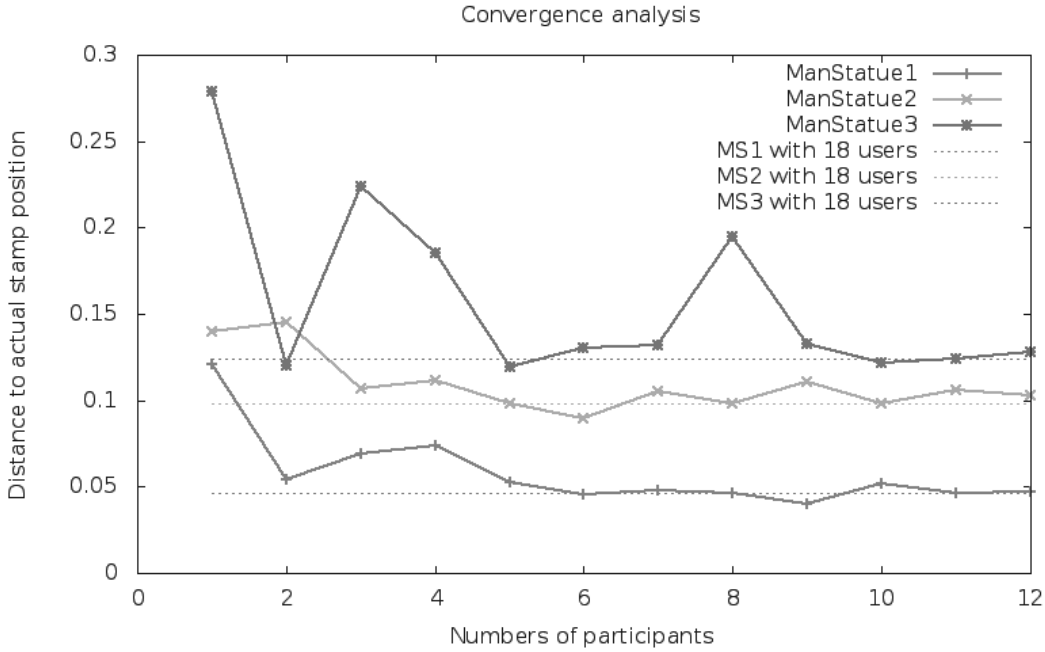


Figure 3.28: Correctness of the center position, depending on the number of users taken to compute the recommendations.

So, we select n random users out of 18 users, then compute the recommendations using only the interactions of those n users. The output of our method is a 3D point (corresponding to a mode out of the Mean-Shift algorithm). We can then compute the distance between this 3D point and

the actual 3D position of the stamp (see figure 3.28). By taking a high number of permutations and averaging the distances we get a point on the diagram.
The results show that the crowd could be converged around 10 users in order to get the same precision for the recommendations as we have with 18 users (represented by the horizontal lines).

3.4.2.4 Crowdsourcing 3D interaction is possible and useful!

This first experience shows that analyzing 3D crowd navigation to identify ROIs and give recommendations for the next generations of users is possible, even with a small members of users. We also show that the recommendations are useful for subsequent users.
In the next section, we see a different way to ease 3D navigation by creating semantic links between 3D models and rich content.

3.4.3 Enhancing online 3D products through crowdsourcing

We propose a second approach on easing 3D navigation by building semantic links between a product's textual description and its corresponding 3D visualization. These links help gathering knowledge about a product and ease browsing its 3D model. Our goal is to support the common behavior that when reading a textual information of a product, users naturally imagine how it looks like in real life.
Here again, we use crowdsourcing to generate an association between a textual description and a 3D feature. A user study of 82 people assesses the usefulness of the association for subsequent users, both for correctness and efficiency. Users are asked to perform the identification of features on 3D models; from the traces, associations leading to recommended views are derived. This information (recommended view) is proposed to subsequent users for performing the same task. Whereas the associations could be simply given by an expert, crowdsourcing offers advantages: we have inexpensive experts in the crowd as well as a natural access to users' (eg. customers') preferences and opinions.
We have shown in the previous session that browsing online 3D object is slow. Thus, 3D interaction is cumbersome due to the numerous degrees of freedom in 3D interactions.
In our work, we consider 3D product pages as shown in Figure 3.29. Initially, a typical product page simply combines basic text (on the left) and a browsable 3D model of the product (on the right). Our framework aims at enhancing the product presentation and produces a richer multimedia content including semantic association and links between the text and the 3D visualization. In the following of the paper, we will argue that outsourcing this simple multimedia edition job is valuable: we have inexpensive experts in the crowd as well as a natural access to users (e.g. customers) preferences and opinions.

3.4.3.1 Proposed approach

As in the previous experiments, we aim at easing 3D navigation by giving recommendations. This time however, the recommendation is linked to a textual description of the product. Our goal is to guide the user to interact with familiar parts of the products in less time and more easily: user can readily locate a feature defined by a textual description using the proposed semantic association. Moreover, feedback is given on the relevance of the proposed recommended views. First, we show that the enhancement of 3D models is appreciated by users: a majority of users assess that the recommendation was useful for them, and browsing a single 3D object using mouse interactions and the proposed links from the text is preferred from

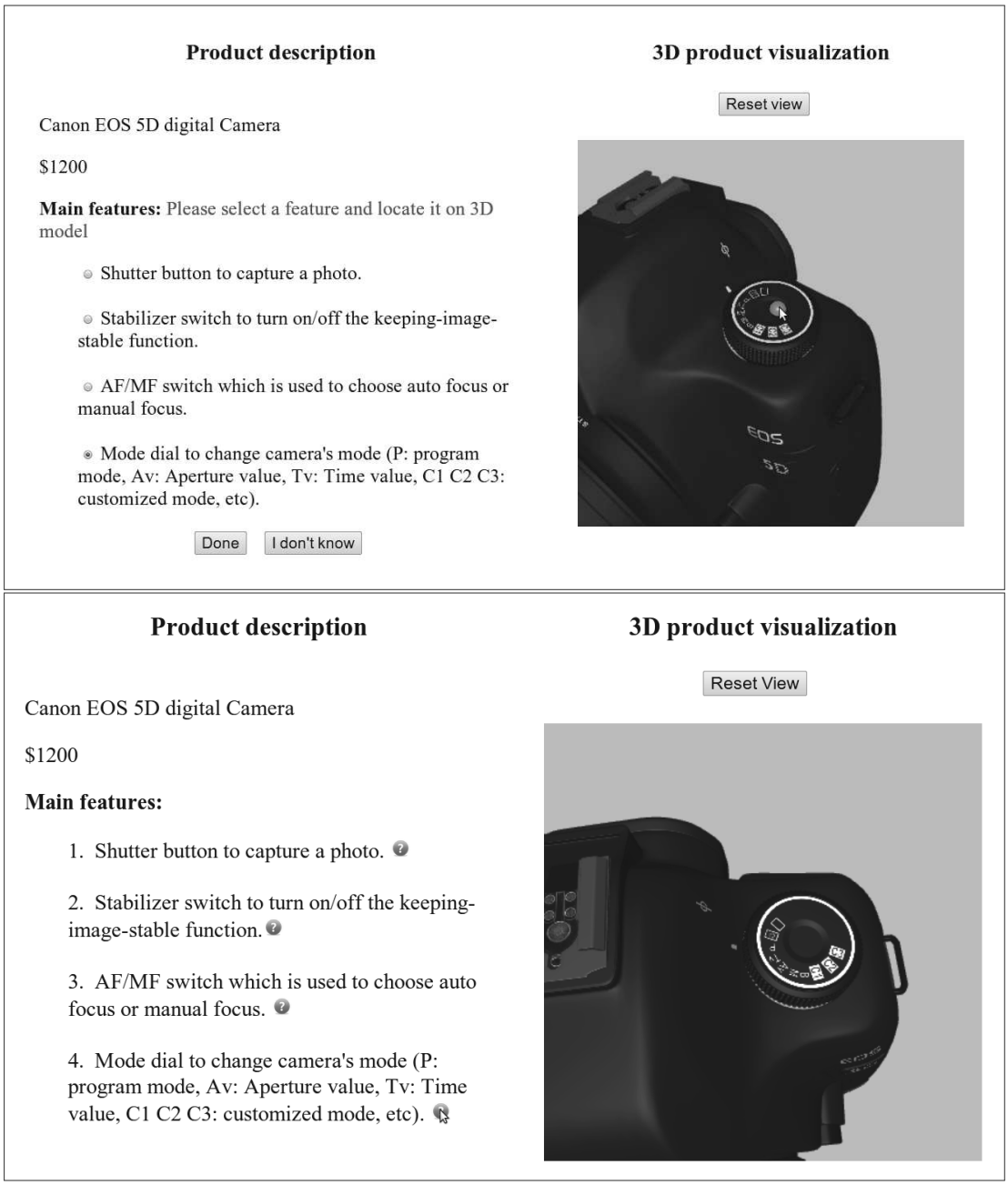


Figure 3.29: A screenshot of the proposed interfaces

having only mouse interactions. Second, the enhanced 3D leads to better performance in terms of efficiency (the amount of time required to perform a given task). In terms of correctness, we shall see that wrong recommendation generate more wrong answers, however, by including user feedback on the usefulness of the recommendation we were able to decrease the number of wrong answers compared to the original test (no recommendations).

Finally, we wonder if this semantic association can be created by the seller or an expert hired by the seller. A product description may contain many features, up to 15 or 20, and an e-commerce website can have thousands of product in all types of real-life objects from house furniture to

working devices. Moreover, hiring an expert to create semantic associations for an enormous number of features is expensive and suggestions given by experts, usually paid by sellers, are biased towards advertising the products. We show that the semantic associations may be derived from crowd-sourcing. We show that the lack of expertise for the crowd source may be overcome: we have implicit and explicit clues to assess the quality of a recommendation. Implicit clues come from the distribution of answers of multiples users. Convergent answers, that is, a set of samples well approximated by their average/mean value correspond to features easy to identify whereas features difficult to detect lead to few or sparsely distributed answers. Explicit feedback on how useful a recommendation was, have also been collected and given to following users to help them assess if a recommendation could be trusted. By getting opinion of users on the derived association, we manage to improve the quality of their answers.

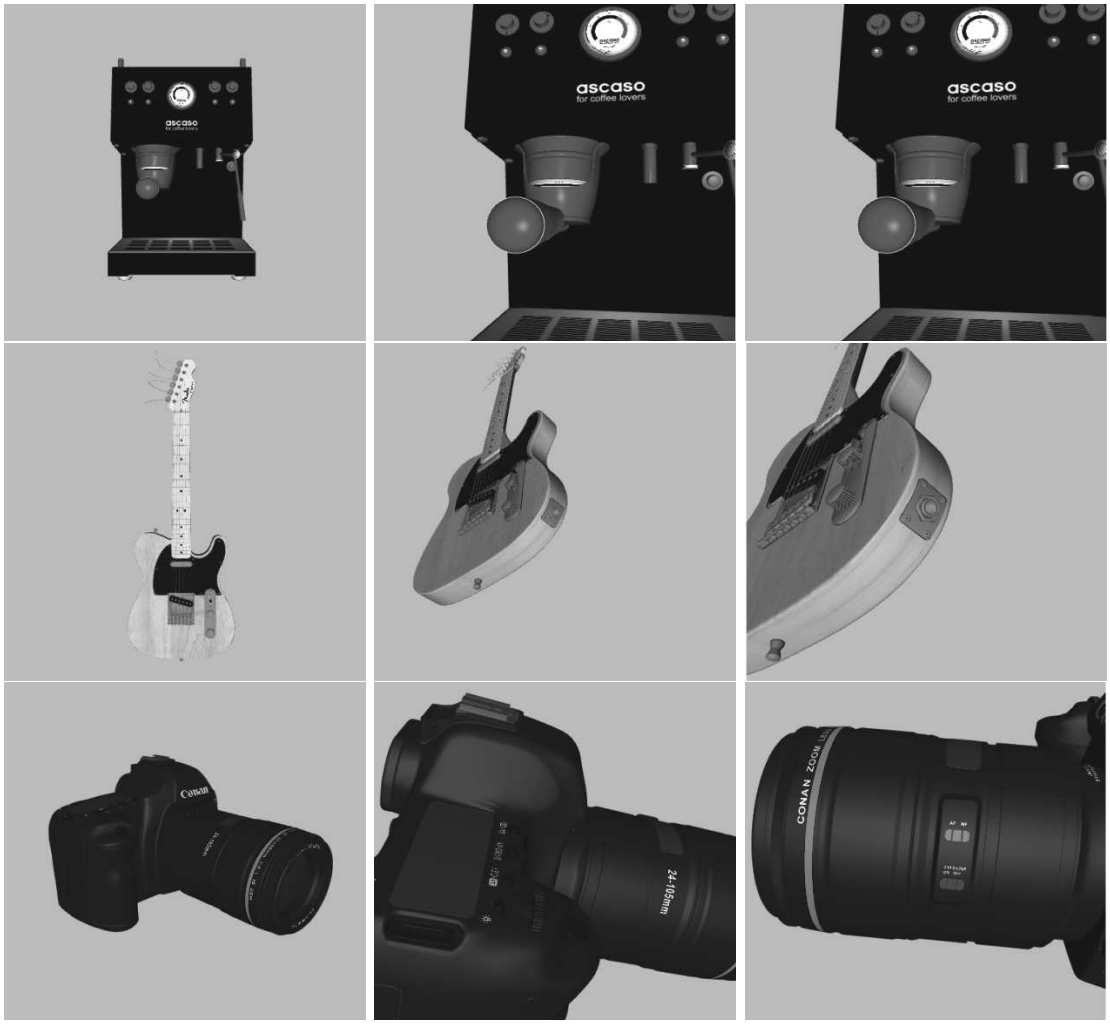


Figure 3.30: The original views (left column – part1). Recommendations generated from crowd-sourced associations (middle column – part2). Recommendations improved by providing opinion on the usefulness (right column – part3).

The following section details the setup of the four conducted experiments.

3.4.3.2 Experiments

Experimental setup
Participants Participants were volunteers and the experiments took place in presence of an organizer. We avoided remote online experiments to insure the relevance of recorded time. A total of 47 male and 35 female participants aged from 19 to 40 (mean 27), mostly from the university community took part in the experiment. None of the users participate in more than one part of the test.
Models. Since we target the e-commerce use case, we chose models of everyday life in our user study. The models are of different complexity, aesthetics, and require various knowledge from daily life to technical aspect or specific domain. More specifically, we use 6 models, including two cameras, two guitars and two coffee machines. Products have been chosen for having both technical features and aesthetical relevance. For each model a list of four features defined by a textual description is given (see Figure 3.29).
Platform The experiments were conducted on a web-based platform, in order to reproduce the conditions of a real e-commerce website. Here also we use the x3dom framework. While users were completing their tasks, we collected their traces and stored them into our database. The data collected consisted in:

- The amount of time it took the user to find each feature's visualization in 3D.
- The world co-ordinates of the 3D *marked-point* on the surface of 3D object, representing the feature's recognized position. World co-ordinates of normal vector at this look-at point and its camera position are also logged.
- Time and events (room/pan/rotate/double click, etc) created by users.
- "I don't know" events if users cannot locate the feature.
- Level of user knowledge about each product.

Protocol of the user study The user study consists in 4 different parts, each of which evaluating a different aspect of the work. Every user from part1-3 watched a tutorial video explaining what to do during the user study. Then each of these users performed a test, in order to get familiar with the interface and 3D browsing. Finally, users completed a task, slightly different at each step of the study, on all the test models.

- Part 1: *Association.* For each model, the user selects features one by one and tries to locate the feature on the 3D object. If he/she is able to find it they can double-click on it and a red dot marks the location on the object. We call this red dot the *marked-point.* If the user is not able to locate a feature, he/she can click the "I don't know" button and go on to the next feature. We then ask users to estimate their degree of familiarity with the product, ranging from 1 (novice user) to 5 (expert user).
- Part 2: *Evaluation.* In this step users have to do the same operations as in Part 1, except that each time users choose a feature, a recommended view is automatically displayed to suggest the corresponding 3D visualization of the feature. Then, for each feature, we ask the user if the recommended view was helpful or not.
- Part 3: *Helpfulness Evaluation.* Users are in the same conditions as in Part 2 except that they are given the results of the usefulness evaluation of the recommended views from Part 2. We still ask them to do the same operations as in Part 1.

- Part 4: *Novel interface Evaluation*. In this step, users are asked to interact with two different interfaces: one is just a 3D product along with its textual description, and the other one, though very similar, associates a view on the 3D model to each feature. We then asked users which of the interface they would prefer in an e-commerce scenario.

Recommended View Generation

We model the *marked-point*s of the crowds to present the recognized feature position in 3D model. There are many methods to indentify a feature from a set of *marked-point*s. In our case, since logged 3D points are on the surface of the objects, we chose the common maked-point as the point from the set the closest the median *marked-point*s (each coordinates considered separately). Since the median is a robust operator, outlier corresponding to unconcentrated, uncompetant or malicious users are not disturbing the common *marked-point*. The variance, is also computed and quantifies the dispersion level of *marked-point*s. Figure 3.30 shows examples of the recommendations we get from our system.

We see in the following session how we shall use the variance to characterize the feature.

Results and interpretation

Quality of answers and recommendations

The recommendations (Figure 3.30) are generated by analyzing traces from users that participated to Part 1 of the experiment. These views are associated to one feature from the textual description. The views should show the filter-holder of the coffee machine, the jack socket of the electric guitar and the stabilizer switch of the photo camera. The coffee machine is a simple and popular object and almost everybody knows what a filter-holder is, which explains why the recommendation based on expert users is identical to the recommendation based on all users. The guitar though is a more specific object, and the user needs to know some technical details about electric guitars to be able to locate the jack socket. For such objects, we observe that the recommendation based on expert users is more accurate than the recommendation based on all users. Finally, the stabilizer switch of the camera is difficult to find and some users misidentified it. However expert users correctly identified the position of the switch, which results in a completely different recommendation than the one based on all users. These examples are representative of all the other features. Among the 24 features (6 3D objects with 4 features each) that were studied in our experiments, we can group the recommendations obtained into three classes by analyzing the variance among answers, and the difference between answers depending on the user (self estimated) expertise.

- Easy features: lot of good answers, both by experts and others users, small variance; 18 out of 24 features. Good recommendations.
- Technical features: lot of "I don't know" answers mostly by non-expert users, large variance; 4 out of 24 features. Imprecise recommendations.
- Hard features: lot of wrong answers from non-expert users, large variance. 2 out 24 features. Bad recommendations.

Efficiency to execute the task

Figure 3.31 presents the distribution of the average time taken by users to locate the features on the 3D models. On this figure, we focus on the 3D models that appear on figure 3.30: a coffee machine, an electric guitar and a camera. We plot the average time taken by user to execute the task (correctly or not) which is to click on the position of the feature on the 3D object. For each

	Right	Do not know	Wrong
Part 1	75%	12%	13%
Part 2	80%	12%	8%

Table 3.4: Percentage of right answers, "I don't know answers", and wrong answers to the tasks for Parts 1 and 2a of the experiments.

product (e.g. the camera) there are two sets of results corresponding to the average time of users from Part 1 and Part 2 of the experiment. First, note that compared to the previous experiments (where users were asked to find a stamp), the time it takes to locate a feature is more reasonable (less than 30 seconds in all cases). It is clear when looking at this graph that the users from Part 2, who were given recommendations based on traces of users from Part 1, are quicker to perform the task than users from Part 1. Table 3.4 shows that users not only perform the task in less time, but the percentage of good answers is slightly higher in Part 2 (80%) than in Part 1 (75%).

Influence of the comments from previous users

Having presented the three types of features (easy, technical, and hard) and shown that recommendations help users being more efficient in completing the task, we now want to establish the interest of asking users to evaluate a recommendation's usefulness. This is the purpose of Part 3 of the user study.

	Easy	Technical	Hard
Helpfulness	84%	55%	25%

Table 3.5: Percentage of users from Part 2 thinking the recommendation was helpful, for three features each characteristic of one class.

Table 3.5 presents the percentage of users from Part 2 that found the recommendations (generating from traces of users from Part 1) helpful to complete the task. We show these percentages for three particular features, each representative of one the three classes we defined earlier : easy, technical and hard. The easy feature is the "Steam button" on one of the coffee machines, the technical feature is a "microphone" from one of the electric guitar and the hard feature is the "Stabilizer switch" from one of the cameras. We could sum up the results from Table 3.5 by saying that users from Part 2 find the recommendation for the easy feature very helpful, the recommendation for the technical feature moderately helpful and the recommendation for the hard feature not helpful.

Feature class	Part 1		Part 2		Part 3	
	Wrong	DNK	Wrong	DNK	Wrong	DNK
Easy	10%	0%	0%	0%	0%	0%
Technical	15%	40%	0%	50%	0%	35%
Hard	35%	35%	43%	35%	35%	50%

Table 3.6: Percentage of wrong answers and "I don't know" (DNK) answers on three representative features for Part1, Part2 and Part3 of the user study.

Table 3.6 presents the percentage of wrong answers and "I don't know" answers (the percentage of right answers can be easily deduced from these numbers) on the three features previously

mentioned, for the three first parts of the user study. For the easy feature, 90% of the users from Part 1 accurately locate the feature on the 3D model which results on a good recommendation. Therefore when presented with the recommendation, all users answer correctly to the task of locating the feature. For the technical feature, users from Part 1 have more trouble identifying the feature on the 3D model. 40% of users acknowledge they do not know the answer, and 15% of the users locate the feature at the wrong place. This means there are still 45% of users that identify the feature's location and it enables our system to produce a meaningful recommendation. The recommendation, when provided to users from Part 2, indeed prevents them from locating the feature at the wrong position, but increases the number of users acknowledging they do not know where the feature stands on the 3D model. This result shows that users somehow trust the recommendations enough to prevent them from answering badly, but do not trust it completely to prevent them from answering that they do not know where the feature is located. Users from Part 3 however, because they also have information about the helpfulness of the recommendation, are more incline to trust the recommendations and we observe that 65% of them get the correct localization for the feature which is a great improvement compared to users from Part 1 (35% of good answers) and Part 2 (50% of right answers).
For the hard feature, 35% of users do not know where to locate the feature and 35% of the users locate it wrongly. The generated recommendation is therefore really bad, and users from Part 2 perform even worse. 43% of them position the feature at the wrong place. The helpfulness measure being very low (25%, see Table 3.5) 50% users from Part 3 prefer to answer they do not know the place of the feature.

Qualitative evaluation
Finally we asked the last group of users (from Part 4) to compare two interfaces and decide which one they prefer. We presented them, for each 3D model, with two different interfaces. Both interfaces include a 3D visualization of a product and a textual description of the product. But one of the interfaces is interactive, and for each feature in the textual description a button can be clicked to automatically change the view of the product, displaying the recommendations generated thanks to traces of users from Part 1. Figure 3.29 shows an example of this interactive interface, and on this example the user has just clicked on the 4th feature's button.
Figure 3.32 presents the results of this study. Users tend to prefer the interactive interface (with recommendations) for all the products, even for "camera2" in which 2 recommendations among 4 are incorrect.

3.4.3.3 Interpretation

The conducted experiment and user study has shown two particular points: first, the proposed enhancement of the 3D interactive product has proved useful in two ways: qualitatively, users have appreciated it, quantitatively, it has improved their performances. We showed that the proposed 3D enhanced multimedia content is valued by users. Quantitatively, the proposed framework has also improved performances of users: the time required to achieve a task is much shorter when using recommended views. Also, accuracy is improved: more right answers are given in part 2 and 3 than in part 1. This is true in particular for easy features, which most of them are, but not necessarily for technical or hard features. However, for technical features, we have shown that filtering users considering themselves as expert does help.
Finally, the crowdsourcing approach needs to be defended with respect to simply asking a reliable expert to create the association. An easy argument is the cost: convincing customers to provide

some feedback by giving them advantages seems possible, whereas paying a professional expert may be more expensive. Beside the cost, the expert is going to be biased towards its employer, most likely the seller. Another argument is that the identification of easy, technical and hard features has been possible through traces analysis (in particular the dispersion of the answers, and the difference between expert and non expert users). This grouping of features has been important since the results were very dependent on the nature of the feature. Finally, we have seen that opinion on the recommended views has proved a very valuable information: for hard features, giving a feedback on the usefulness of the recommended view did decrease the number of wrong answers (more people did 'admit' they did not know). In some sense, it tells us that if the crowdsourced information is not perfect, and certainly worst than an expert advice, a further iteration of crowdsourcing can derive a valuable confidence measure on this information.

3.4.4 Conclusion, limitations and perspectives

We have shown here that crowdsourcing techniques may apply to interactions with a 3D object. This is an very first but important step that opens the door to further developments. In the different experiments we have successfully enhanced the navigation of new users from the traces of previous users. We have proposed two different interfaces that significantly simplify the interactions of users with the online 3D object. The second experiment shows that working on adaptive recommendations is also relevant. Recommendations were improved: (a) by feedback on their usefulness, (b) by considering self-evaluated expertise of users. We propose to further (c) estimate (through the variance of the answers) the nature (easy, technical, hard) of features since the collected data seem to be strongly correlated to this nature.

These first steps in crowdsourcing for 3D multimedia content are encouraging. Among lessons learned fr this first test beds, we also showed that interacting with 3D takes time: in the first experiment, (non assisted) users took almost one minute to locate one stamp, in the second experiment, they took more than a minute to locate the four features on the object. How long is a regular person ready to spend on a 3D model on internet? Easing or simplifying navigation seems not only a good idea, it seems necessary for proposing 3D as a usable multimedia content.

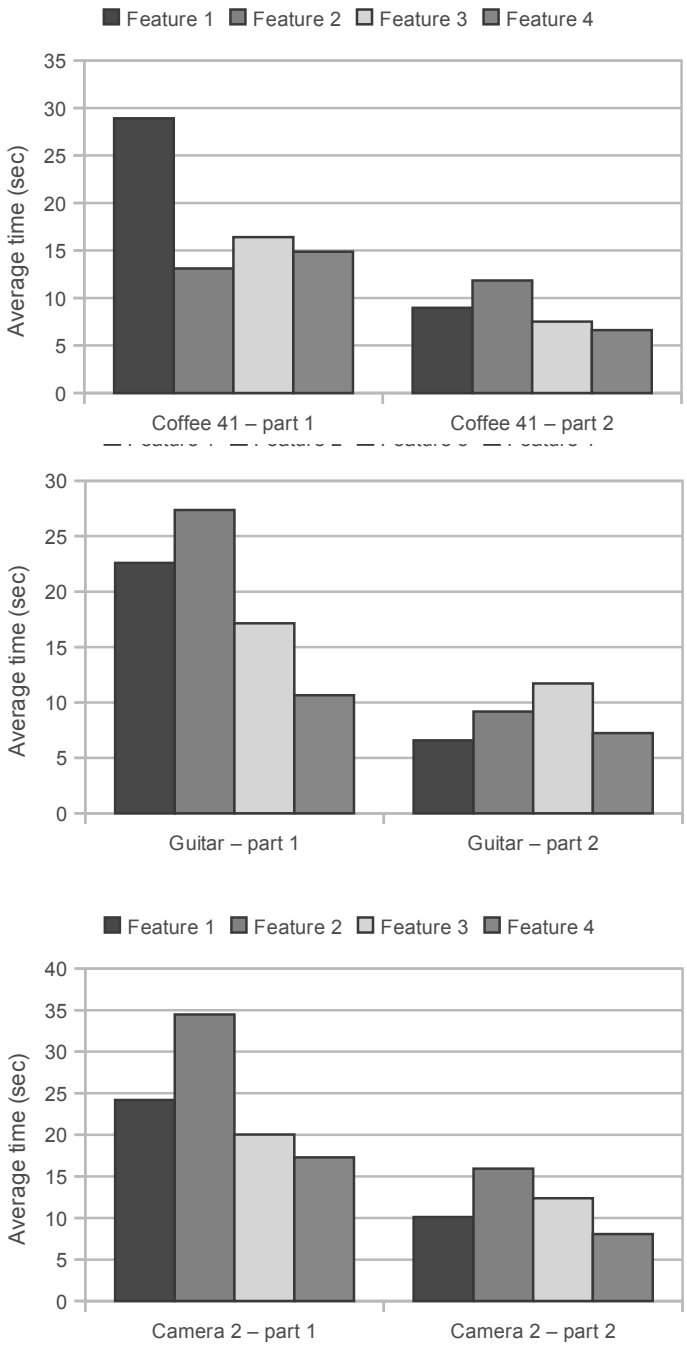


Figure 3.31: Average time (in seconds) to locate the features in 3D

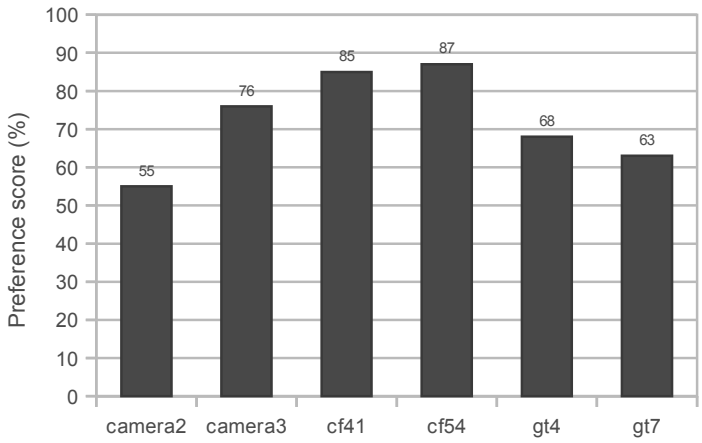


Figure 3.32: Evaluation of user preference to the recommended interface

3.5 Conclusion, limitations and perspectives

In this section we have proposed three contributions for easing the manipulation of 3D content. Similarly to the first chapter where reconstruction was based on images, the tracking is based on 2D($+t$) content. The second application identifies partial similarities within a 3D parametric model. A perspective could be to fill the gap between these two applications: it could be very relevant to be able to identify a given 3D object appearing in a video. This would open the door to indexing video content with 3D data as an input. Of course, this problem is not new, as 3D shape recognition is an old field (e.g. [Besl 1986]). However, there has been much more work on one hand on image registration (e.g. see [Zitova 2003] for a survey); and on the other hand on 3D shape registration (e.g. see [Tangelder 2008] for a survey). Indexing videos with 3D objects could benefit from the recent bibliography on the identification of partial similarities between 3D objects. Also, videos, like images, are much easier to generate than 3D models. So videos could be a good resource for the edition of a existing 3D model.

From the work in the last session, we have seen that interacting with 3D is still a cumbersome task. Indeed, many articles from 2008 or around can be found, announcing 3D coming into e-commerce websites. Five years later, the number of websites offering 3D content has not increased much. Whereas 3D seems to be attractive in movie theaters or in the game industry, 3D content is still not as used in the customer market. We believe that actual interactions with 3D content maybe one of the reasons why 3D content is still difficult to use. Another possible reason is the latency involved when downloading 3D models. The next chapter will propose solutions to this problem.

Chapter 4

3D Compression and Transmission

Contents

4.1 Motivation: remote access to 3D content

We showed in the previous sessions that 3D content is being increasingly present as multimedia content. We have shown and developed techniques to create and manipulate 3D content efficiently. In this session, we will address the following problem: streaming remote 3D data.
Different applications provide online 3D content that is being accessed by remote clients. The first of these applications are virtual museums. Providing access to virtual art pieces has many advantages. First, it allows to visit a museum without needing to travel. The most popular project that aimed at creating 3D virtual museum pieces, at least in the scientific community, is the Stanford's Digital Michelangelo Project [Levoy 2000] that digitized statues made by Michelangelo. Similar projects include the Digital Sculpture Project[1] or the project to digitize Rodin's sculptures [Miyakazi 2006]. Second, virtual models provides an access to art pieces that may be too delicate to be exposed to the public. This is the case for the Gargas cave, a prehistorical site where painting dating from 25 000 BC that can not host too many visitors in order to preserve the intergrity of the site. The *sanctuaire des mains*, a part of the cave that is now close to the public, has been digitized to provide a virtual visit in the adjoint museum[2].
Another advantage is that virtual models can be viewed from any view points. It is particularly interesting to see the David statue from a *face-to-face* view point, as shown in Figure 4.1. We can even be shown in a virtual scene portraying the context a museum piece used to be in its contemporary time. The amount of data constituting a high-quality 3D model can be huge. For example, the statue of David, from the Digital Michelangelo Project, consists of 2 billion polygons. After lossless compression, the total data size is 32GB.
A second set of applications containing large online 3D content are NVE (networked virtual environments). For example, online games offer virtual 3D content created usually by professional content providers that needs to be accessed by remote clients. Other online virtual environments, like Second Life or *activeworlds*[3], is driven mostly by user-created 3D objects which are downloaded on demand as users explore the virtual world. For cultural heritage, ancient cities have been rebuild. For example, a project reconstructed a virtual copy of Rome, some centuries BC[4]. A more general framework, the European project *3D-CONFORM* aims at developing further tools for easing the digitizing of 3D content and associated annotations in the context of cultural heritage. Also, virtual 3D copies of the modern real world are progressing: both Apple's Maps, the map application for iPhones, or Google Earth [5] now include 3D viewing. The content is either reconstructed from 3D imagery, or contributed by users. There are nowdays around 150 cities available in Google Earth. There is a real rush on who will provide first some 3D model of our world.
Another set of applications to use online 3D content is e-commerce. The website `http://p3d.in` proposes a general tool for providing/sharing online 3D models. Toyota[6] offers the possibility to configure a car with option and then visualize it in 3D. *La redoute*, a clothes shop also proposes to dress a virtual model[7]. Note however, that the main website of this shop remains a classical website with only photos. In both these 3D online application, the downloading time of the 3D

[1] `http://www.digitalsculpture.org/`

[2] nesplori@ is an exhibit proposed to the visitors, before or after the cave visit, that offers a numeric and interactive exploration of the Gargas cave `http://www.numerigrottes-pyrenees.fr/p-nestploria_fr.htm`

[3] `http://www.activeworlds.com`

[4] `http://romereborn.frischerconsulting.com`: for the project page ; `http://vimeo.com/32038695`: for a virtual tour ï£ijï£ijof Rome in 320 BC

[5] `earth.google.com`

[6] `http://www.toyota.fr/cars/new_cars/configurator-index.tmex`

[7] `http://www.laredoute.fr/espace-je-cree-mon-look.aspx#` "be your own stylist"

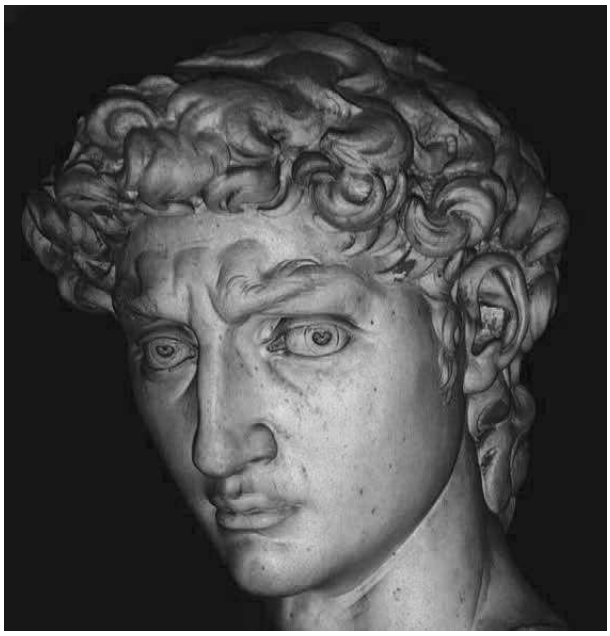


Figure 4.1: The head of the David from Michelangelo: one can see his strange eyebrows and forehead, and his diverging gaze. These particular assets optimized the beauty of the large statue (the body is more than 4 meters high) seen from someone standing next to its pedestal. An interview of Marc Levoy about the diverging gaze of David is available at `http://graphics.stanford.edu/projects/mich/publicity/npr-atc-13jun00/npr-atc-13jun00.WAV`

content takes time (a significant latency).
So, a first challenge is to organize the online 3D data in order to provide access to numerous clients at the same time. For transmission to the clients, the data needs to be as compact as possible, so compression algorithms for 3D content need to be derived. However, providing the content is not the only requirement; for these virtual 3D environments, the question of system requirements for the client, in terms of resources, is very relevant. The challenge is to be able to provide access to this very rich 3D content to numerous clients, including light clients, that is, users with light devices (e.g. a cell phone) and a limited access (e.g. a wireless connection). As accesses happen from different platforms (heterogeneous clients, accessing through networks of varying capacities), the model needs to be adaptive. We want the content to be available at multiple resolution, sent in a compact representation, and insure that important data will be received by the client as early as possible, even in case of a unreliable network.
In the first section (4.2), we motivate the proposed specific framework to stream 3D content over a possibly unreliable network, looking at the characteristic of multiresolution or progressive 3D models. As we propose to use progressive meshes for 3D meshes, we recall the basics. As meshes are not suited for modeling plants, section 4.3 develops a compact and progressive representation for plant models. Experiments shows that this progressive representation offers a good compression rate. We also adapt the proposed model for streaming to very light clients (on a mobile phone). Then, section 4.4 sets a generic framework for transmitting the 3D content, illustrated on progressive 3D meshes, and the progressive plant representation over a lossy network. The two following sections present application of the 3D streaming framework: section 4.5 studies *3D preview streaming*, whose goal is to download a 3D model while the user gets a video-like preview of the model; section 4.6 proposes specific approach for very light

clients. Finally, we conclude and consider the perspectives.

The work presented here witnesses a long time on going collaboration with Wei Tsang Ooi from NUS (National University of Singapore). The section 4.3 has been developed in the context of the Ph.D. of Sébastien Mondet whom I co-advised with Romulus Grigoras; The plant modeling part was a collaboration with the CIRAD from Montpellier, in particular with Frédéric Boudon. More details can be found in Sébastien's dissertation [Mondet 2009a], and in the related papers [Mondet 2007, Mondet 2008, Mondet 2009b]. The section 4.4 was developed in the context of the Ph.D. of Wei Chen from NUS, and has led to the common publication [Cheng 2007]. Section 4.5 is part of Shanghong Zhao's Ph.D. (NUS) work and published in [Zhao 2013]. Section 4.6 relates a short paper of Andra Doran (Master student in Toulouse) [Doran 2009], and the work of Minhui Zhu, Ph.D. in NUS who I advised as she spent a year in Toulouse in 2012-2013. The discussed work corresponds to the paper [Zhu 2011].

4.2 Streaming 3D: a specific framework?

Generally, there are two main methods for accessing content available remotely across a network. The first one downloading a file, followed by its usage (visualization, computation, etc.). Download is the base of the Internet: for example, web pages are downloaded through the HTTP protocol (*HyperText Transfer Protocol*) before being rendered/presented to the user, e-mails are exchanged by SMTP (*Simple Mail Transfer Protocol*) servers by file-downloading, music and movies are massively exchanged through downloading. The second method is streaming. Streaming multimedia consists in constantly presenting the media to an end-user while it is being delivered/transmitted. The first world-wide success of streaming, were the internet radios, for example, the SHOUTcast service which has been using HTTP for internet audio broadcast for more than 10 years. Now video streaming systems are also common place. Streaming media allows to access more or less interactively very large content, progressively; without waiting for download.

In our case, as presented in the introduction of this chapter, 3D scenes are very large content and 3D walk-through is a highly interactive application. Progressive streaming is thus a natural way of accessing 3D objects in these kind of applications. Considering the state of today's networks, streaming media is a challenging task. Internet is based on a best effort network of networks, lacking any Quality of Service (QoS) guarantees: bandwidth is variable and there are random packet losses and desequencing. Therefore to ensure reliability and quality of service while dealing with these characteristics, end-to-end application-level mechanisms are needed. With the years, things have got obviously better; but problems remain, for example on wireless and/or mobile networks. Moreover, applications requirements are increasingly demanding, for example, for video streaming, applications need smaller start-up delays, smaller channel-switching delays or even multi-channel streaming.

To handle the requirements of the networked applications (streaming-based or not), the transport layer is dominated by TCP (*Transmission Control Protocol*), and UDP (*User Datagram Protocol*). TCP is mostly used for downloading and less-interactive streaming, and UDP is used for highly interactive and real-time applications. The fragmentation and packing process, together with the scheduling of the packets, is generally called packetization. The operating system can take care of bare packetization in the case of TCP (it is actually the default behavior). But for UDP and/or applications which require fine-tunned performance, packetization needs to be tackled at the application level, i.e. while being aware of the characteristics of the

transmitted data. This is particularly the case for the streaming of 3D objects, since 3D models have a particular dependency structure –as we see in the next section.

4.2.1 Characteristics of 3D data

In this chapter, we consider that the input data is one or several 3D objects that need to be streamed. As mentionned in the introduction, considering multiresolution models is natural. Multiresolution coding have been proposed for classical model representations, like for triangle meshes (e.g. [Alliez 2001, Alliez 2005, Taubin 1999, Devillers 2000]), for point-based surfaces (e.g. [Pauly 2003a, Rusinkiewicz 2000, Kobbelt 2004, Fleishman 2003]) or for hybrid representations e.g. [Chen 2001]. This coding lead to defining different level of details, and this details conrrespond to interdependent pieces of data. The dependencies between their elementary pieces of data (triangle, points) are usually local, since they are limited by the geometry. For example, for subdivision surfaces a vertex at a given level of detail depends only on nearby vertices from the previous level of detail, given by the support of the subdivision mask. More generally, the dependencies on multiresolution 3D can be represented, in a generic way, by a DAG (Direct Acyclic Graph). If A and B are geometric object to be send in packets, an edge in the graph from A to B models the fact that B depends on A, in other words, that to decode B we need to have decoded A. The main characteristic of these decoding dependencies is that coarser resolutions are important to decode finer ones. Hence, in this whole chapter, we require 3D data to be organized as interdependent binary chunks, following a partial order (defined by the DAG and therefore induced by the dependencies).
Video streaming has been very weel studied. However, we shall see that significant differences exist between multiresolution 3D data and video structures so that using video streaming schemes for 3D is not feasible. First, in video streaming, every packet should be received in time, or it will not be played back. Hence, generally sending new data is more important than resending old data. On the contrary, in streaming of a progressive 3D model, old data (lower resolution) is usually more important than new one (higher resolution), since the reconstruction begins with the most significant refinement. Given this observation, retransmission of a lost packet is not only useful, but it should also take priority over transmission of new packets. Second, in progressive 3D data, the order of between the binary chunks is only partial. No complete order between the chunks exists, unlike in video where frames must be displayed in sequence. A new geometric chunk can be rendered immediately as long as all the parts it depends on have been received. When a packet is lost, the subsequent received packets may still be decodable. But the chunks received afterwards that depend on the lost packet, however, have to wait until the lost packet is retransmitted successfully before they can be displayed. This observation hinted that we should reduce the dependencies between packets as much as possible, since dependencies cause delay in rendering details. Third, a video frame is usually larger than a packet, causing the streaming application to split the video frame into several packets. We shall see that it is not the case for the considered progressive 3D representation.

4.2.2 Compression and transmission of 3D meshes

Mesh compression algorithms have been using specific traversal techniques to efficiently code the topology of the mesh [Deering 1995, Rossignac 1999]. Touma and Gotsman has proposed an approach relying on the fact that most vertices are regular [Touma 1998]. [Bajaj 1999, Lee 2009] have proposed compression schemes taking into account some attributes on the vertices. Further improvements of the compression rates have been proposed by [Alliez 2001, Isenburg 2003]. A

survey of these methods is given by Peng [Peng 2005]. An alternative approach improves the regularity of the mesh by the object with semi-regular grids (Roudet and Payan give a survey on these methods [Roudet 2011]).
In the following of this section, we consider progressive meshes. The advantage of this progressive representation is to decompose the progressiveness in very fine grain elements (vertex splits). Also, the proposed streaming method can be adapted to any kind of progressive representation, the only requirement being that the dependencies are modeled by a DAG (direct acyclic graph). Whereas the papers proposing mesh compression have considered the transmission of their 3D data, they restrict themselves to lossless transmission. In our setting, we do consider the eventuality of losses.

4.2.3 Progressive meshes

Progressive transmission and rendering of 3D objects requires multiresolution representations of data. One such representation, progressive mesh, has been proposed by Hoppe [Hoppe 1996]. The technique is based on an operation called edge collapse, and its reverse operation, vertex split (see figure 4.2).

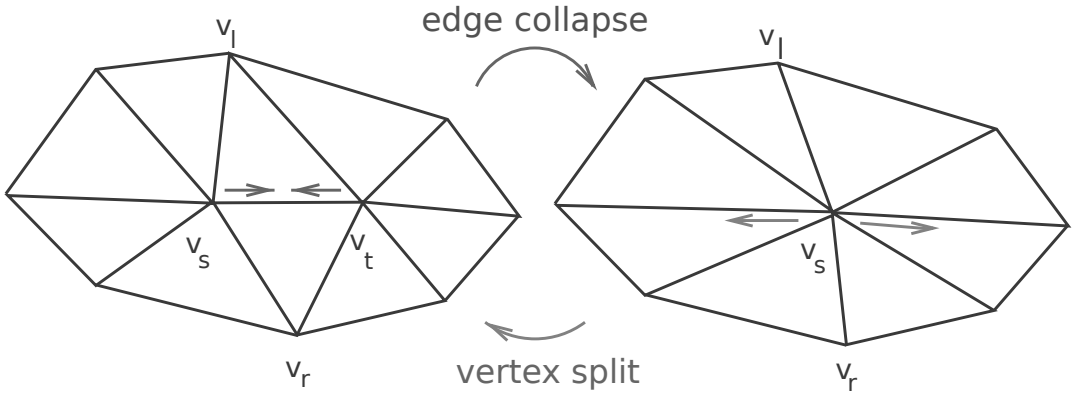


Figure 4.2: The basic opposite operations in progressive meshes: a vertex split and an edge collapse.

Given a (nonprogressive) 3D mesh, the technique applies a series of edge collapses, simplifying the model by reducing the number of vertices and faces. The final, simplified model obtained after this process becomes the base model. Given a base model, we can reconstruct the original model by reversing the edge collapse operations through vertex splits, incrementally adding new vertices and faces. So, a progressive mesh can be represented by the base model and a series of vertex splits. Moreover, there are dependencies between vertex splits and the base model, as well as among the vertex splits. A vertex split operation might need a vertex or a face created by another vertex split as input. Figure 4.3 illustrate the dependency graph structure among vertex slits. Note that there may not be any cycles in the dependency graph.
Progressive meshes are well adapted for streaming since they offer the finest level of progressivity; it allows refinement at the granularity of vertices. This progressivity is crucial for our application since a streamed vertex split only depends on the vertex splits that generated its neighbors, and not on other refinement operations of the same level (like for subdivision surfaces). Therefore, only dependencies among the vertex splits need to be considered.

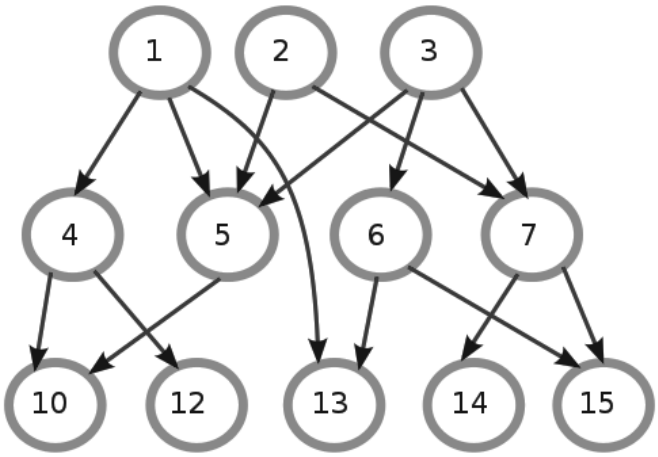


Figure 4.3: Dependencies between vertex splits: the structure is a DAG (direct acyclic graph).

Many extensions and variations to Hoppe's progressive mesh have been proposed (e.g., [Pajarola 2000, Cohen-Or 1999, Alliez 2001, Chen 2003]). The main idea behind these extensions is to combine multiple operations into one, thereby further reducing the redundancy and improving the efficiency. In particular, Pajarola and Rossignac proposed a Compressed Progressive Mesh (CPM) representation to reduce the size of a progressive mesh [Pajarola 2000]. While our work only considers the original progressive mesh, our model is general enough to model many of these extensions.
Before resuming discussion on 3D streaming, in section 4.2, we first develop a progressive representation for plant models in the next section. Then, section 4.4 how to stream progressive 3D data, like progressive meshes or progressive plant models, providing an importance metric for the different part of the model.

4.3 A compact and progressive representation for plants

As we will show in the next section (4.3.1), meshes are not adapted for plant modeling. We thus want to derive an adapted progressive representation for plants. As our choice for meshes was progressive meshes in order to have a fine level of progressiveness, we will like also to derive a progressive representation for plants. This rest of this section presents an original compact and progressive model for plants.

4.3.1 Previous work on compact plant models

We have already modeled plants in the first chapter, and quite naturally, we do use here a similar model of plants. However, we first justify why meshes, or progressive meshes, can not be considered for plant modeling. Realistic and detailed plant mesh models can require up to hundreds of thousands of polygons. Remolar et al. [Remolar 2002] estimated that a plant generated by XFrog, a well known plant modeling platform, can consists of 50,000 polygons to represent the branches. The plants can have 20,000 or more leaves, which themselves consists of several polygons. Neubert et al. [Neubert 2007] reported the plant models that they used consists of up to 555,000 polygons. These numbers are for a single plant. Moreover, due to

the particular topology progressive meshes degrade strongly the coherence of a plant under simplication, as shown in Figure 4.4.

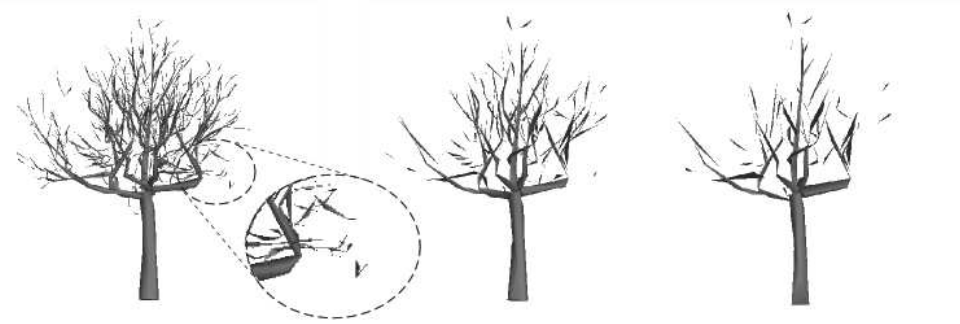


Figure 4.4: A walnut tree modeled by a mesh, and progressively simplified. We see that the topology of the tree is not preserved.

Some alternative representations are based on billboards i.e. pre-rendered images used as impostors, e.g.. [Meyer 2001, Decaudin 2004, Behrendt 2005]. Other representations are based on points (e.g. [Weber 1995, Deussen 2005]). Both billboard and points representations rather focus on foliage (leaves); thus they can be used as complementary to ours since they are usually complemented with polygonal representations for the trunk and the branches. By default, however, they seem more dedicated to static representations: they have to be attached to a skeleton representation to support animation.
Next, we present the full resolution representation we started with.

4.3.2 Base representation

We have alrady seen in previous sessions that plants branching systems are most often represented as a connected set of generalized cylinders. Here also, we consider such a representation. We still define the branching system as a set of connected parametric curves with control points. These topological structure representations have the advantage of being compact compared to more discrete representations such as mesh and provide support for animation (which is not the case of the simplified models whose connectivity is lost in Figure 4.4). For example, the Walnut of Figure 4.5 at full resolution only requires about 10 772 control points using generalized cylinders compared to 278 632 triangles using a mesh model. By default, however, the representation based on a connected set of generalized cylinders is not adapted for compact and progressive description. Our goal in this section is precisely to fill this gap.
The branches are organized inside a n-ary tree data structure modeling the structure of the plant. We call such a data structure a *n-tree*, to avoid confusion with the concrete plant object we are actually modeling. Our representation focuses on the branching structure of a plant and is thus based on a skeletal representation. Each branch is a generalized cylinder: the axis curve is a 3D Bézier curve defined by its control points, as show in Figure 4.6. Axial parameters such as the radius, color or texture coordinates can be defined as Bézier curves along the branch. In practice, we use for now only radii as axial parameters, defined by 2D Bézier curves. The branches are organized inside an n-ary tree data structure defining the structure of the plant. The root of the n-tree is the trunk of the plant and branches borne by the trunk are the n-tree children of this trunk. Each child branch contains a attachment parameter ($u \in [0, 1]$) giving the position of the attachment point on its bearing parent branch (as in [Prusinkiewicz 2001]).

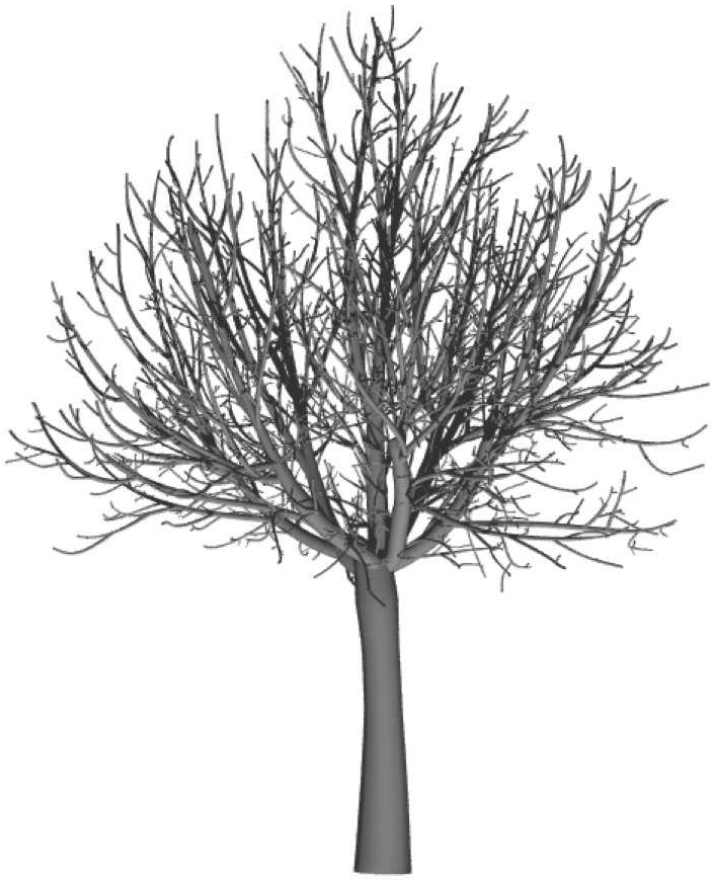


Figure 4.5: The Walnut model (digitized by Sinoquet et al. [Sinoquet 1997]).

The parameter u defines the first control point of the Bézier curve of the child branch. The remaining control points are encoded in the child branch by their three coordinates in space (left of Figure 4.7). Axial parameters are also defined thanks to the attachment parameter u. We take the example of the radius but the scheme could be adapted to textures or colors. The case of the radius of the branch illustrates how attributes along the branch are coded. A radius is defined as a positive real value along the branch. To model it as a smooth function along the branch, we represent its values as a series of control points (u_i, r_i) of a Bézier curve of degree m, where (u_i), $i = 0 \dots m$ is an increasing sequence in the interval $[0, 1]$ that defines the location of the branch, and (r_i), $i = 0 \dots m$ characterizes the radius for the corresponding given location (right of Figure 4.7). Note that the degree of the radius curve is not related to the degree of its bearing branch (it is usually much lower).

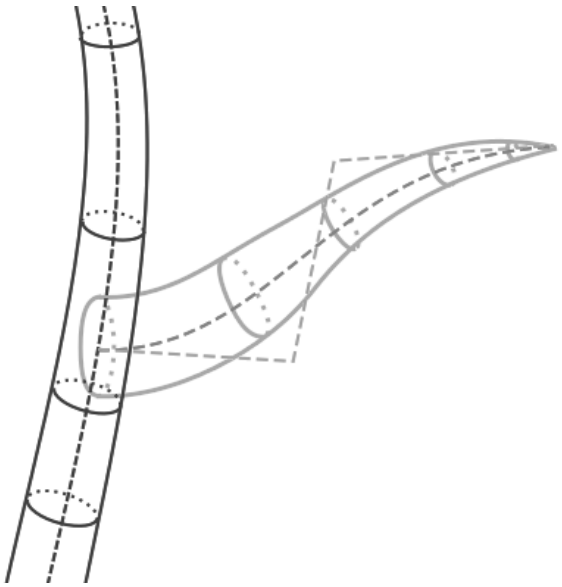


Figure 4.6: Representation of a branch as a generalized cylinder.

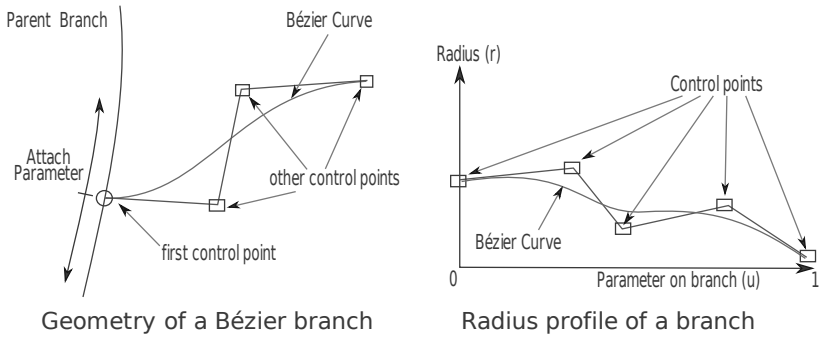


Figure 4.7: Representation of a branch: on the left the axis curve, which first point is a point of its parent branch given by a parameter value u; on the right, the Bézier curve representing the radius along a branch.

4.3.3 Compressing the structure: overview

Our starting point is a natural scene using plant models based on the skeletal representation given in the previous paragraph. We want to have a progressive representation of the plants, in order for a client accessing the scene remotely to be able to progressively decode and visualize the plants in the scene. Figure 4.8 outlines the steps from encoding to streaming of our representation, and guides the presentation of this section. One main concern is to propose a representation where at low resolution, the plant would already look realistic: for interactive application, as a user may quickly pass by an object without waiting for the complete model to be downloaded, the low resolution of the model is very important . The n-tree data structure already has a natural progressive nature since it introduces a hierarchy corresponding to the natural tree hierarchy of branches. However, if we follow this hierarchy, a low resolution model would only include the main branches, leading the a low resolution very different, for example in terms of density, than the complete plant model. We therefore rather chose a compressed, progressive representation that allows having a good density, even at low resolution, by decorrelating information into three components called *branch models* (4.3.4.2, *instances*, and *detail vectors* (4.3.4.3).

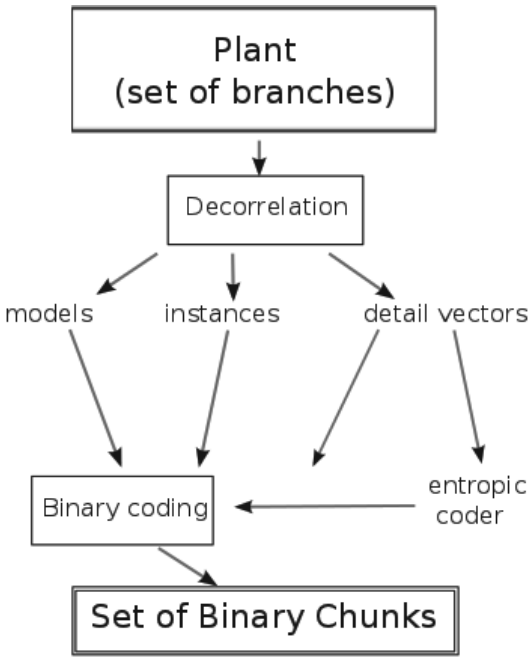


Figure 4.8: Overview of the coding process.

4.3.4 Decorrelation

To encode a plant as a compressed progressive representation, we exploit the similarity of the Bézier curves representing the branches and the radii. The idea of the compression algorithm is to replace the absolute coding of the control points by differences compared to an average Bézier curve. We group the branches and the radii independently to profit from the similarity inside each group of Bézier curves. As the differences are small, they may be coded on fewer bits, leading to a compact coding. A simplified overview of this decorrelation process is shown in Figure 4.9 for the case of Bézier curves representing branches (the process for radii is equivalent but less visual). First, we group the curves following a similarity criteria (section 4.3.4.1). Then,

we apply a normalization transformation to the curves of the group (section 4.3.4.2) and extract a model Bézier curve, that is, an average curve which bet approximate the curves of the group. This model curve allows us to express each Bézier curve of the group as two entities: instances and details (section 4.3.4.3). The instances depend on the normalization parameters and allow the decoder to instantiate the model Bézier curves to build approximated branches and radii in its within the tree. The details are the differences between the actual curves and the approximate one; adding the detail vectors moves back the instantiated model branch to the original branch.

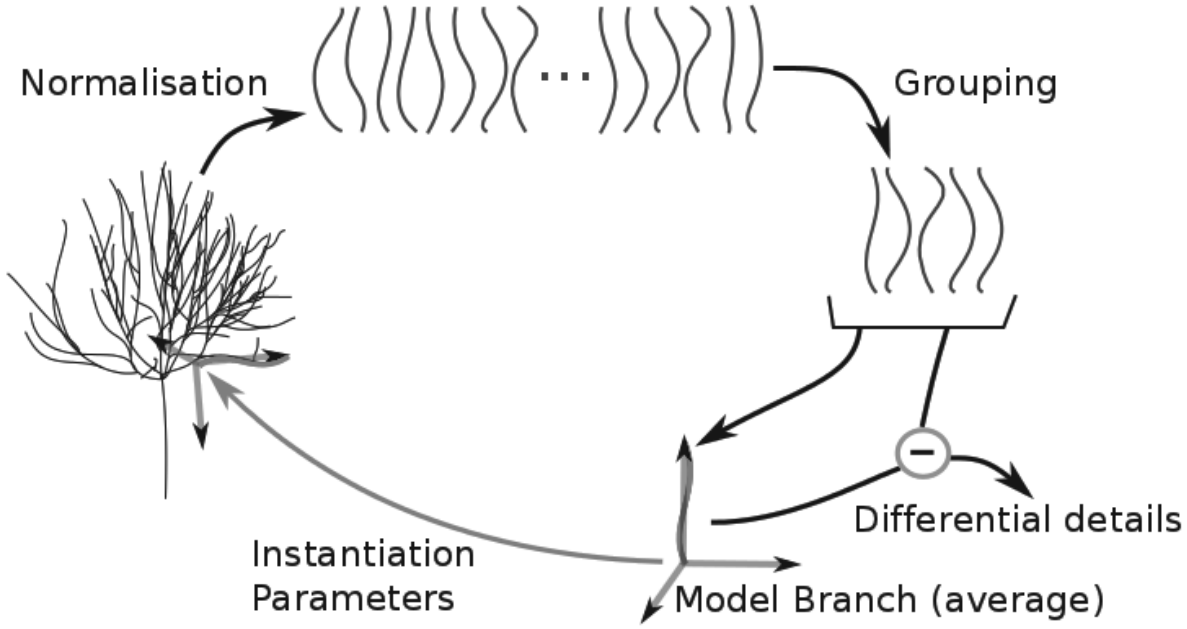


Figure 4.9: Overview of the decorrelation process.

4.3.4.1 Grouping branches

The first step in the decorrelation process is to group the branches, or Bézier axis curves and radii curves, according to a similarity criteria. The accuracy of the approximation by model Bézier curves as well as the performances of the entropy coding of the detail vectors may depend on the quality of the grouping. We have implemented several grouping filters, each to satisfy different criteria: compression efficiency, quantization error minimization, or the visual aspect of the progressive decoding. A complete description and comparisons of grouping policies are detailed in [Mondet 2009b]. These experiments lead us to choose a best compromise setup, which consists in using the degree reduction option for both branches and radii, and grouping only the branches using the scale-based filter (creating four groups). Obviously our choice can be challenged: other performance criteria or other experimental data (plant models) may produce a different best compromise. However, if changes in grouping strategy induce small quantitative changes, they do not perturb qualitative observations on the results. Here we only detail the two grouping strategies that led to the so-called *best compromise* compression scheme that we use in the experiments (section 4.3.6).

Degree Reduction

In a first scheme [Mondet 2008], we had grouped Bézier curve by degree, that is, by the number of control points, simply to be able to compare their control points. However, to have Bézier curves comparable independently of their degree, we build a standard representation by preprocessing the curve: we use a degree reduction algorithm (c.f. Figure 4.10). A Bézier curve of degree bigger than 2 is approximated by a curve of degree 2. We apply the algorithm called `CEQ 2` proposed by Bogacki et al. [Bogacki 1995] which is based on Constrained Equioscillation and has the property of interpolating the endpoints of the approximated curve (which is essential for us due to preserve normalized curve, as we show in the next section 4.3.4.2).

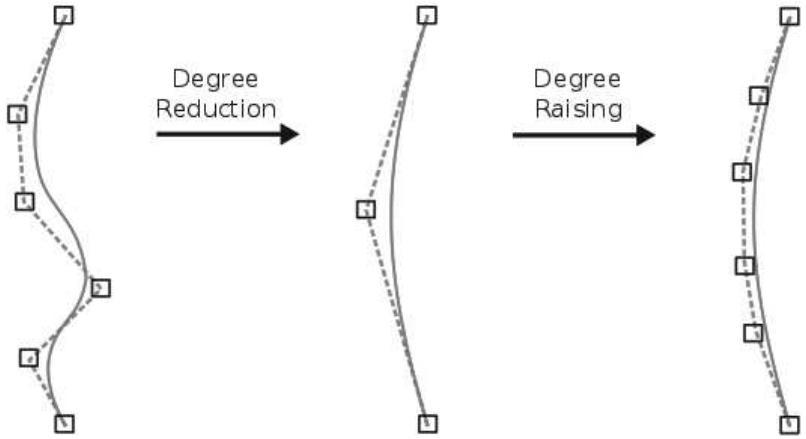


Figure 4.10: On the left, an original curve, in the middle a degree 2 curve after applying the degree reduction, on the right the same degree 2 curve with as many control points as the original curve (applying the classical degree raising procedure).

Scale-Based Grouping

An additional grouping strategy, called *scale-based*, uses the length of the branch (or the average radius for radii curves) to group the curves. We group the Bézier curves by partitioning the set of line segments joining the first and the last control point. We uniformly partition the interval defined by the minimal and maximal lengths (or scales) into a chosen number of ranges. Then, we create the groups by associating the curves with the range containing its length. This grouping strategy has been designed for Bézier curves representing branches (even if it is usable for radii). As a typical tree has fewer long branches and more short branches, longer branches tend to be grouped in smaller groups, while shorter branches are grouped into bigger groups. Since short branches are likely not to bear children branches, having a less accurate version of these branches in the intermediate tree is visually acceptable. For long branches, the shape of a branch affects all children branches and therefore affects the overall shape of the tree. Thus, moving a long branch also causes popping effects, which we want to avoid. For this reason, this scale-based grouping gives better visual results for the progressiveness of the tree, but it also provides good compression results(c.f. section 4.3.6).

4.3.4.2 Normalizing and defining *model* branches

In order to compare and to code differences between two branches, a so-called *standard form* of the Bézier curves is proposed.

Transformation of Branches
We make all branches comparable thanks to an affine transformation converts back and forth between an original branch and its *standard form*. The affine transformation is defined so that the first and last control points of the original Bézier curve, map to the origin $(0, 0, 0)$ and the point $(0, 0, 1)$ respectively (c.f. Figure 4.11). We characterize this first mapping by a translation,

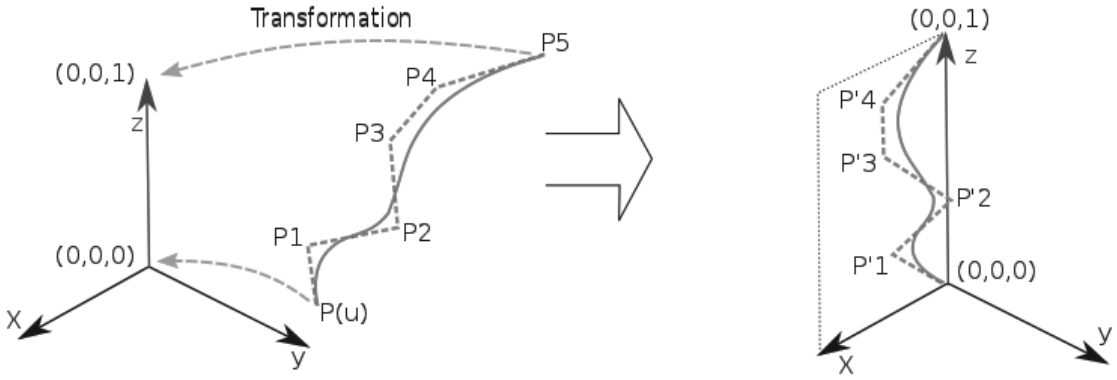


Figure 4.11: A branch axis curve modeled by a Bézier curve is mapped into its standard form: the end control points are mapped on the z-axis and the barycenter of the remaining control points lies in the plane $x - z$.

two rotation angles, and a uniform scaling factor. Since we choose to apply a uniform scaling, there is a remaining degree of freedom, which corresponds to another rotation around the z axis. To completely define the affine transformation, we fix the rotation around the z axis so that the center of gravity (or average) of the remaining control points, lies in the (x, z) half-plane. In the case of degree 2 curves, which have only one remaining point (the two other points have been fixed to $(0, 0, 0)$ and $(0, 0, 1)$), this latest rotation brings all curves totally in the same half-plane.

Transformation of Radii
Since the parameters u_i already belong to the interval $[0, 1]$, we normalize the family r_i by dividing it by the average norm of the radii. All normalized radii profiles provide hence the same average thickness.

Choosing the Model Curves
The previous process generates a set of groups containing normalized representations of the Bézier curves. We now compute the model curve for each group as an average of the other curves. In this document, as we have applied degree reduction for the Bézier curves of degree greater than 2, the average Bézier curve has degree 1 or 2 (and in the case of branches, its endpoints are also $(0, 0, 0)$ and $(0, 0, 1)$). The remaining control points of the model curve are computed such that the i^{th} control point of the model curve is the barycenter of the i^{th} control points of the curves of the group.

4.3.4.3 Instances and Details

We decompose now a branch of the tree into two parts, the *instance*, placing correctly the model branch in the tree, and the *details*, to recover the original branch. The instance of a branch is an approximation of the branch, placed in the tree, at the right place in terms of topology, but not quite at the right (geometric) location. To draw an *instance* of a branch on its parent branch cylinder, we need:

- a reference to the model branch;
- a reference to the parent branch;
- the attachment parameter (u);
- the inverse of the (affine) normalization transformation.

For radii the case is simpler, the requirements are:

- a reference to the model radius;
- the scaling factor used during normalization.

All those *instantiation parameters* define what we call *instances* in our modeling scheme. The encoding of an instance branch cylinder is now defined by five entities: the branch model, the radius model, the instantiation parameters, the branch details and the radius details. Next we define the *detail vectors* to express the original curves and radius relatively to the models.
For each original branch and attached radius in a group, we have the corresponding *instance*. We now need to restore the remaining information, *detail vectors*, to recover the original branch. Since degree reduction has been used for normalization, all model curve are curves of degree 1 or 2. We raise the degree of the model curve before computing the detail vectors (Figure 4.10). Degree raising is a deterministic algorithm (see for example [Farin 2002a]), it can be therefore used both on encoder and decoder side, and provide the same result. We now code in differential form the corresponding Bézier curve relatively to the model curve, storing, instead of the coordinates of the control points, their differences to the corresponding control point of the model curve (c.f. Figure 4.12). Those differences are the *detail vectors*.
A branch is now defined by its *instantiation parameters* and its *detail vectors*.

Progressive representation of the plant
The proposed representation allows branches of a plant to be displayed progressively as generalized cylinders in two ways (see Figure 4.13). First, the natural hierarchy of the n-tree is followed. The model branches are transformed thanks to the instantiation parameters; the resulting instances are displayed attached to their parent branch, showing an approximate cylinder for the branch. The approximation of a branch is built by applying the inverse of the normalization transformation to the model curve. Second, the detail vectors are taken into account and may refine the shape of the branch previously rendered.
In terms of dependencies within the representation of the plant: if we exclude the header chunk, the interdependency can be observed from the references and from the decodability, e.g. to decode an instance one first needs to have decoded its parent branch and its branch and radius models. There are two main families of dependencies: topological dependencies and those generated by the differential coding. The first family is related to the n-tree structure of the plant: a given cylinder depends on the parent branch it attaches to. The second family includes the

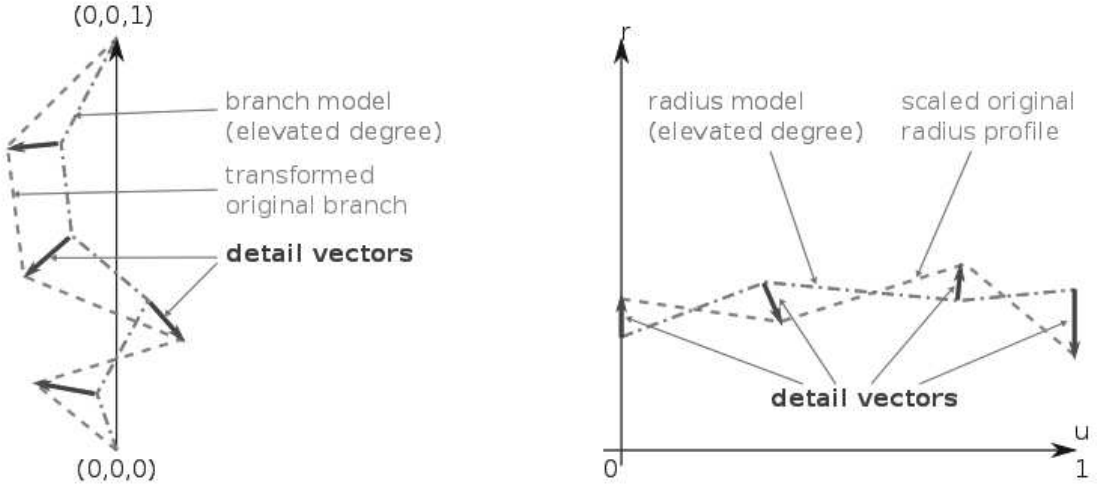


Figure 4.12: The detail vectors are the difference between the control points of a curve of degree n in normalized form and its model curve (with degree raised to n).

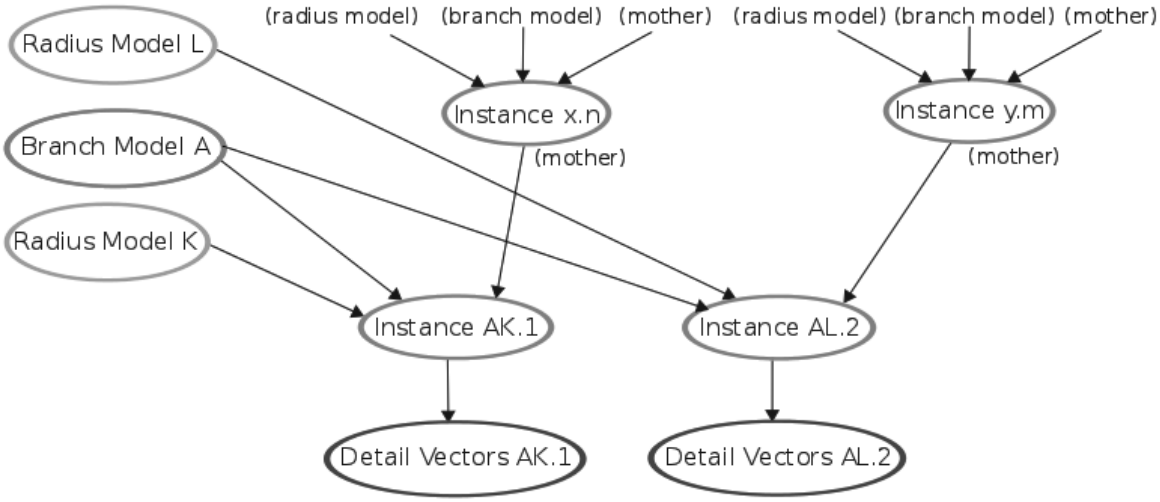


Figure 4.13: An example showing dependencies between different components of the progressive representation. Instance AK.1 depends on its parent instance x.n and instance AL.2 depends on its parent instance y.m; those are topological dependencies. Detail vectors AL.2 and AK.1 depend respectively on their corresponding instances. Instance AL.2 also depends on a branch model A and on a radius model L, and instance AK.1 depends on the same branch model A and on a different radius model K.

dependencies due to differential coding, that is, induced by the progressive coding: on one hand the dependence between a cylinder and its branch and radius models, and on the other hand the dependence between a set of detail vectors and its corresponding instance. Note that a child instance is independent of the detail vectors of its parent instance; in section 4.4.1 we use this independence and show how we prioritize between children and detail vectors.

4.3.5 Binary coding

After transforming the base representation, a set of connected Bézier cylinders, into a progressive representation, we obtain three classes of data: *models*, *instances* and *details*. We now code them to build a set of interdependent pieces of data, called *binary chunks*. For each chunk, general information, is stored in a *header*.

4.3.5.1 Data contained in each class

We detail for each class, *models*, *instances* and *details*, the data that need to be sent.

The Models
First, for the model branches are in normalized form, so the first and last control points are always $(0,0,0)$ and $(0,0,1)$, so they do not need to be coded. Only intermediate control points need to be defined. As we have used the degree reduction option, a branch model can only be of degree 1 or 2, therefore only 0 or 1 control point need to be coded. To reference both the branch model and radius model while decoding an instance, we need to define a model identifier; a positive integer strictly smaller than the total number of models.

The Instances
To instantiate a model branch on the progressively decoded tree, we first need to reference its parent branch, which is another instance. This requires coding of an instance identifier and a reference to another instance. Both identifiers are also bounded integers. Then to place the curve on its parent branch, we need the attachment parameter, which is a bounded real number ($u \in [0,1]$). Then we need to transform the model curve of the branch. For that, as shown previously, we need first to reference the branch model of the instance by its model identifier. Then we need the normalization transformation. As seen in section 4.3.4.2, the normalization is the composition of one translation, one scaling and three rotations. However, thanks to the attachment parameter we can have the position of the first control point of the curve. We know one point, which is $(0,0,0)$, and its corresponding translated point, given by the parent branch and the attachment parameter u. Therefore we do not need to code the translation; we can obtain it from the starting point. The remaining transformation parameters are three scalars for the angles of rotation and a scalar for the uniform scaling.

The Details
As for the models branches, differential details for curves of degree d require the coding of $d-1$ 3D vectors as they are differences between normalized branches. Moreover, to reference the branch to whom the details belong, we need to join an instance identifier. Similarly, details vectors for Bézier curves representing radii of degree m, consist in $m+1$ 2D vectors and its (radius) model identifier. For a given branch shape differences and radius differences are stored together.

4.3.5.2 Coding of Generic Data

Excluding the detail vectors, which will be studied in the next section, we only have three types of numbers to code: *general scalars*, *bounded scalars* and *bounded integers*.

- *general scalars* are floating point numbers that are not bounded; they can be arbitrarily big or small. Floating point is, for us, the safe *default* encoding, when we do not know enough about the number. They are the control points of the model BeÌĄzier curves and the scale factors of the instances (one for the branch and one for the radius).
- *bounded scalars*: the attachment parameters and the rotation angles, are real values that are bounded and uniformly spread within their bounds. Hence, as the bounding intervals of those scalars can be uniformly sampled, they can be serialized more efficiently with fixed point arithmetic. A binary integer can be represented by a ratio ($[0, 1]$) with respect to the bounding interval. Therefore, we have to choose, for each parameter the precision, i.e. the number of bits used to code the number.
- *bounded integers*: identifiers and references can be coded using a limited number of bits: $ceil(log_2(MaxId))$ where $MaxId$ is the maximal number to code. Identifiers are numbered from 0 to $MaxId - 1$, therefore this coding is optimal.

4.3.5.3 Coding of Differential Details

One advantage of multiresolution differential coding is that the induced differences (*detail vectors*) are small. This provides the ability (i) to quantize small detail vectors with a small number of bits, and (ii) to choose accurate binary representative symbols according to their distribution.

Quantization

The quantization can be vector or scalar. We have carried out experiments using vector quantization in [Mondet 2008] and using scalar quantization [Mondet 2009b]. The results show that scalar quantization performs better. Even though vector quantization is slightly better at reducing the entropy, the gain does not compensate the higher header overhead (for more details see [Mondet 2009a]).

Coding

We chose the number of bits per coordinates $bpc = 6$ for the rest of the experiments to insure the mean error to be less than 0.008 (and maximal error at 0.013) for the *Walnut*, using *best compromise* setup.

In order to appreciate the relative weight of various components, details of the *Walnut* model are shown in Table 4.1. For coding dependencies, only 4 (resp. 11) bits are required to identify each of the 11 models (resp. 1870 instances) nodes. For *Walnut*, the base data size is 3020 bits and the total size is 246660 bits (30833 Bytes). Note also that model nodes represent only 2.5% of the data.

4.3.6 Compression results for plant models

Example plants

Our experimentation are based on two real digitized plant models. Indeed, models generated by L-systems, that are more regular would show outstandingly good performances for our compressing scheme since we can benefit from regularity. It is therefore important to test our scheme our real, irregular data. The two trees are a 20 year old Walnut tree (from [Sinoquet 1997]) and an apple tree (from [Costes 2003]). The walnut tree is 7.5m high and 5.8m wide (c.f. Figure

Type	Number of chunks	Size (bits)			
		min	avg	max	total
Models	7	12	112.57	204	788
Instances	1870	137	137.00	137	256190
Differences	1870	27	50.61	251	94632

Table 4.1: Binary coding: data chunks and their size for *Walnut* coded using our "best compromise" options (Header size: 1150 bits).

4.5). It took two weeks to digitize using a Polhemus Space Fastrack electromagnetic device. We pre-processed it by fitting Bézier curves to a series of digitized points representing branches. Our representation is thus composed of approximately 1900 branches with 6900 control points for the branches and 5800 control points for the radii. The apple tree is 6 year old, 2.8m high and 2m wide and is made of 430 branches, 1350 control points for the branches and 1100 for the radii. To extend our experimental range of models, we have also generated some examples using L-systems [Prusinkiewicz 1990]. For example, we used here a fir-like tree composed of 6 945 branches, 208 354 control points for the branches and 13 900 control points for the radii. Of course, if used in an application, L-systems models would have been surely more efficiently coded and transmitted by sending their generative rules and parameters. But determining generative process of a given tree is not always possible, in particular for measured tree.

Figure 4.14: Three examples used in our experimentations.

In order to appreciate the efficiency of our compressed model we have chosen to compare it with a well-known compression method: bzip2. For that we first concatenate all the binary chunks to a file. We must note that if the goal was file-based compression, we could gain a little more by removing a part of the pointer overhead: when binary chunks are concatenated, instances and models identifiers and details references can be deduced from the order in the file. For instance, in the Walnut model we could remove at least 5145 bytes. We do not perform those optimization as in the next session, we want to stream of packetized plants instead of bare file compression. Those results are shown in Table 4.2. The first row contains the size of a basic serialization of geometry and topology of the connected Bézier curves (with floats and integers coded on 32 bits). The second row shows the performance after compressing the file with *bzip2*. The third

Tree name	Size (Bytes) and compression ratio		
	Basic	Basic + bzip2	Our method
Walnut	143608	84519 (1.70)	44098 (3.26)
Apple tree	28404	16026 (1.77)	9766 (2.91)
L-System (fir)	2666968	2358353 (1.13)	269108 (9.91)

Table 4.2: Comparison of coding performance of three methods: basic binary coding, basic coding compressed with bzip2 and our progressive coding (using the best compromise setup). Size is given in bytes and compression ratio is given w.r.t. the *basic* serialization size.

row shows results for our method for the best compromise normalization and grouping strategy, with 6 bits per detail coordinate.

Results of Table 4.2 show that, for the *best compromise* grouping policy and for six bits per differential coordinate quantization (i. e. $c = 6$) on the *Walnut*, *bzip2* compression applied to the basic coding has a 1.70 compression ratio, whereas our coding method improves it to 3.26.

Figure 4.15 highlights the progressiveness of our compressed model; this figure was done using the best compromise strategy fr grouping the plants.

Figure 4.15: The progressive coding/decoding of the Walnut tree with corresponding compression rates.

4.3.6.1 Adding leaves

Leaves have been added to the model, but unlike children branches, are not attached on the branches. They are rather treated in an independent structure: this independence allows to start rendering leaves, even if the actual bearing branch has not been reached yet at the current partial representation. Moreover it avoids popping effects. This choice has been made because of the visual importance of leaves compared to branches. To render the leaves, we draw a polygon at the specified location and orientation, and we map a texture of the leaf with transparency. Figure 4.16 shows some screen shots of the rendering on a mobile device.

In the following session we will consider the transmission of progressive 3D models, corresponding either to progressive meshes or progressive plant models.

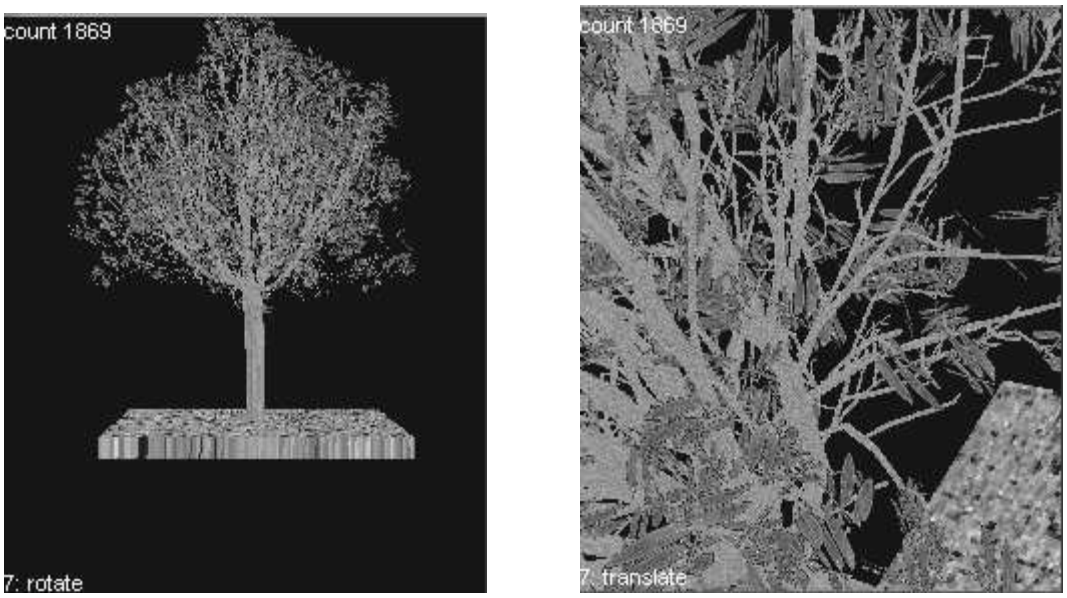


Figure 4.16: A *light* version of the walnut tree including leaves. Leaves are considered independent objects here, to avoid leaves to appear last.

4.3.7 Conclusion, limitations and perspectives

In this section we have presented a compact and progressive model for plants. This model is based on the classical generalized cylinders branching system. The proposed model makes sense for a single plant, but for a forest, generating plants in 3D may be unnecessarily expensive in terms of data size. As mentioned in section 4.3.1, some alternative representations based on billboards or point based models (e.g. [Meyer 2001, Decaudin 2004, Behrendt 2005, Weber 1995, Deussen 2005]) are complementary to ours for a single plant. For a forest, or a relatively far away group of trees, considering a further simplified representation like in [Decaudin 2004] is probably necessary.
The proposed progressive representation for plants branching systems could be also used in compression scheme for more general objects. In the previous chapter, we have studied similarities with a 3D parametric object (section 3.3). Since we are able to detect approximate similarities, we could apply the grouping proposed for branches to similar parts. A model part would represent one of the part, and the other (approximate) similar part, will just need to store the isometry (and the details). This representation would efficiently compress such parametric models. Moreover, since it would provide a progressive coding for parametric models, it would also naturally fit into the streaming framework proposed in the next section.

4.4 Transmission of 3D data

We now have, for meshes or plants models, two progressive representations of our data. As mentioned in section 4.2.1 video streaming is performed on data being totally ordered (by time) whereas 3D progressive representation are only partially ordered (see Figure 4.17). Moreover, video frame dependencies are local in time: the DAG has independent connected components. This induces a main difference between a missing packet for video and 3D streaming: the *persistence* (or durability) of the induced visual artifacts. A missing in a video stream, even

for a key-frame, may have visual consequences only for a few seconds. For a 3D model, a progressive mesh or a progressive generalized cylinders plant, a hole in the geometry can remain visible during the whole walk-through and prevent a lot of data from being decoded. This difference is visible in the dependency graphs of these pieces of data (c.f. Figure 4.17).

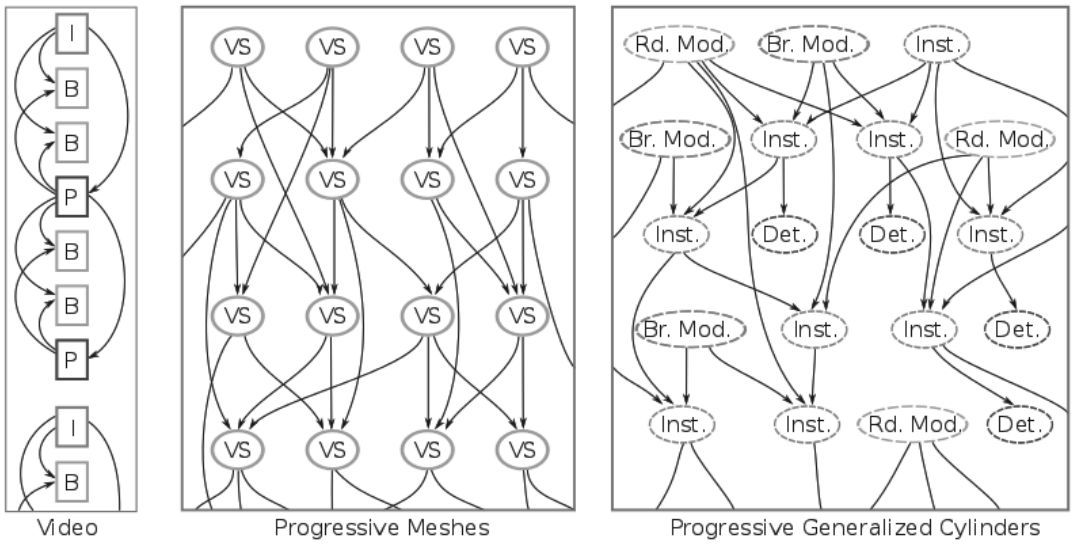


Figure 4.17: Dependency graphs for *IBP-based* video coding (c.f. [Richardson 2003]), for progressive meshes and for our progressive representation for plants.

Given the background on progressive 3D representation, we review the contribution of our work on transmission of these models. We do not detail here the technical content (and in particular the analytic model) but sum up the theoretical and practical results. More details can be found it the related publications [Cheng 2007, Cheng 2011, Mondet 2009b].

4.4.1 Importance of nodes and FIFO sending order

In the following, we consider a generic DAG modeling a progressive 3D model. We call *nodes* the vertices of the DAG corresponding either to *vertex splits* in a progressive mesh, or *models*, *instances* or *detail vectors* in a progressive plant.

4.4.1.1 Node importance and quality metric

In progressive 3D streaming, the quality of the received model increases over time. Because nodes contribute differently to the quality, how quality improves depends on the decoding order of the nodes, which in turn depends on the sending order of these nodes. Because of the flexibility in choosing a sending order, it is possible to choose a sending order so that the quality on the receiver increases as fast as possible. A natural method is to send nodes in the descending order of their contribution to the quality of the received mesh. For that, each node is assigned an importance measuring its contribution to the 3D model. This importance measure needs to be additive and independent of the order in which the nodes are considered. *Additive* means that the quality of the model may be assessed by summing up the importance of the considered nodes. So, one strategy is to always send the most important node first; another strategy is

to packetize the nodes to minimize in order to minimize the dependencies between packets. If there is no packet loss, these two objectives can be achieved simultaneously.
Although intuitively, the importance of the node should be a function of the visual contribution of a node to a rendered model, and therefore depend on the subjective perception of the user, we propose here intrinsic importance for the node, for progressive meshes and progressive plant models.

Progressive meshes
In the validation, the quality metric considered was the split edge length, which satisfies both properties (additive and independent of order). To verify that our model is accurate under different metrics, we also considered the quality with another commonly accepted metric, the Hausdorff distance between the reconstructed mesh and the original mesh. We use the difference of mean face-to-face Hausdorff distance between these two meshes before and after split as the importance of a vertex split.
In the process of creating a progressive mesh, edge collapse are typically done in increasing importance order.

Progressive plants
We propose here a tunable and easy-to-compute quality metric based on geometric considerations. We define the importance for each chunk as follows:

- the importance of a branch model is a constant k_0;
- the importance of an instance is the value of the scaling factor, corresponding to the size of the branch, times a constant k_1;
- the importance of detail vectors is the importance of the corresponding instance multiplied by the average length of the detail vectors (including branches and radii), times a constant k_2;

The constants k_i relate these three metrics to each other. The importance of instances and detail vectors become comparable with the importance of model chunks using the two constants, k_1 and k_2, respectively. It is always better to send model branches quickly, so we always set $k_0 >> k_1$ and k_2. The relative value of k_1 and k_2 can be chosen depending on the application and on subjective criteria. When k_1 becomes larger than k_2, the density of the tree is prioritized. When k_2 becomes larger than k_1, the accuracy of the shape of the branches is privileged. Figure 4.18 illustrates the use of these knobs, with two extreme cases.

4.4.1.2 FIFO strategy

The importance of the binary chunks, computed thanks to a quality metric, allow us to define a total order on the chunks. We define a ordering method which provides a sorting algorithm. While all chunks have not been sorted, we loop on the following steps:

- We retrieve the set of the binary chunks which are decodable at this point, i.e. those whose dependencies have already been sorted/sent;
- Among these chunks, we choose the one with the highest importance to be the next to be sorted.

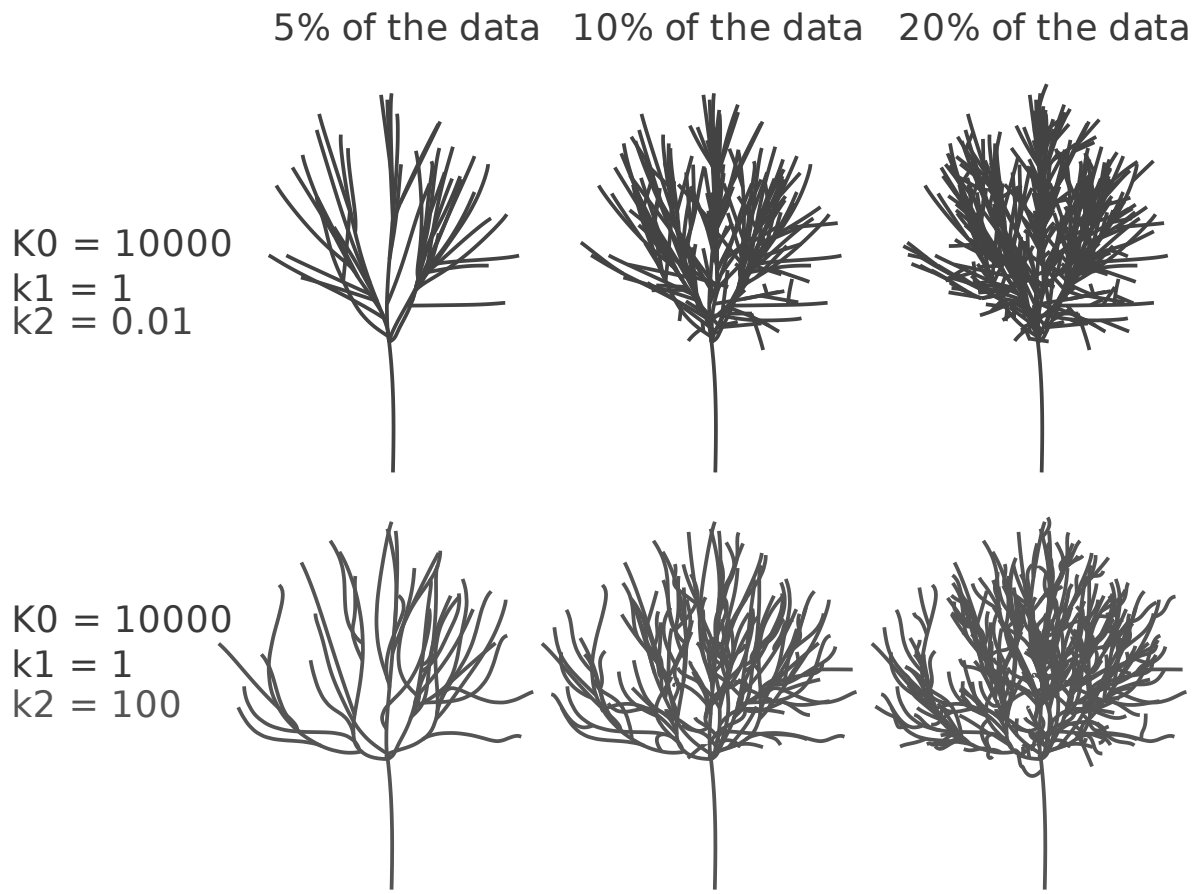


Figure 4.18: The influence of the choice of (k_1, k_2) on the structure of Walnut after decoding 5%, 10% and 20% of the data.

This order leads to the FIFO (First In First Out) strategy. Figure 4.19 shows the progressive rendering of the *Happy Buddha*[8] when using the FIFO strategy.

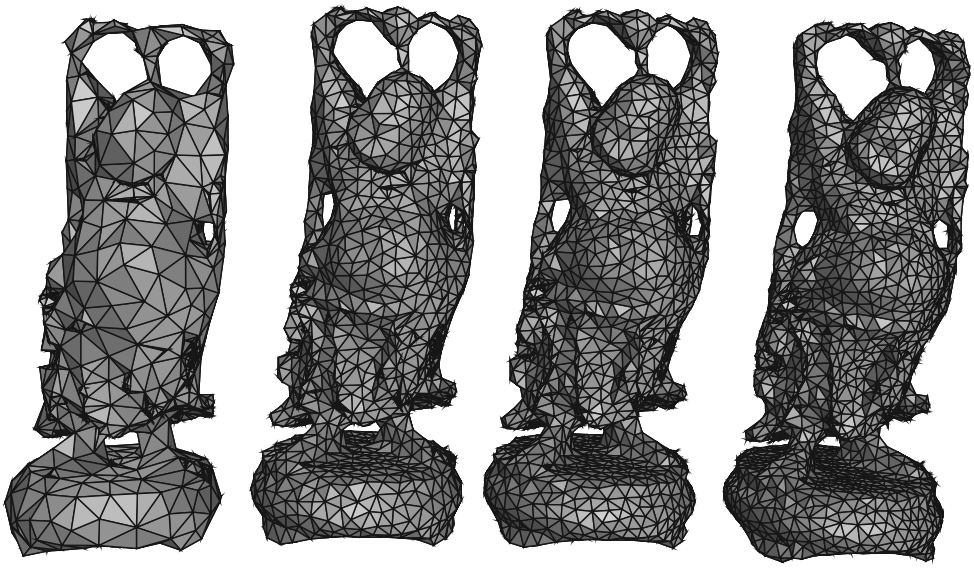


Figure 4.19: The *happy Buddha* partially rendered from left to right, with qualities (from left to right) 2.82, 8.26, 9.25, and 11.44.

Limits of the FIFO strategy

First, note that the FIFO strategy is not globally optimal as shown in Figure 4.20.

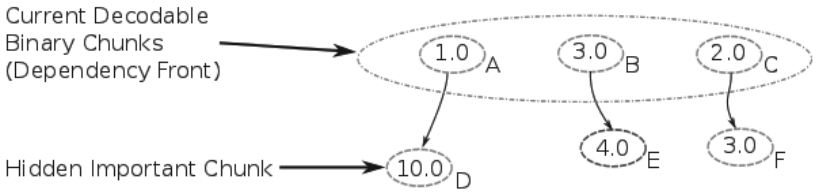


Figure 4.20: An example of non-optimality of the FIFO strategy. The ordering method will give the following order to the binary chunks: $\{B, E, C, F, A, D\}$ but the total quality would have been improved faster with, for example, the order $\{A, D, B, E, C, F.\}$

Moreover, relying only on the importance of vertex splits, is not always optimal when the mesh is transmitted over a lossy network, because dependency also plays an important role in choosing sending order. Considering dependencies, a node's parent is always sent before the node, so a node can be decoded as soon as it is received. But, if a packet loss happens, there is a conflict between resending the lost node, or sending the next important one, that is, between maximizing importance and minimizing dependencies. FIFO maximizes the importance, but so,

[8]from Standford 3D scanning repository `http://www-graphics.stanford.edu/data/3Dscanrep`

may increase dependency among packets. Next section explain how to trade-off between these two objectives.

4.4.2 An Analytical Model for Progressive Mesh Streaming

The descendants of a node cannot be decoded before this node has been decoded. Therefore, when a progressive model is transmitted over a lossy network, a packet loss will delay the decoding of the following nodes if they depend on the lost node; the received nodes cannot be decoded until the lost ancestor node is successfully retransmitted. Hence, another consideration in choosing sending order is to minimize the dependency among packets so that most received vertex splits can be decoded without waiting. Therefore, to choose a proper sending order, it is essential to find a proper trade-off between the importance of vertex and the dependency among packets. Understanding the trade-off requires us to quantify the effect of dependency, which is is non-trivial, as the effect depends on which packets are lost during transmission. Packet losses are random, so the effect of dependency can only be estimated probabilistically.
A first objective has been to quantify the effect of dependency on the rendered quality of progressive models when they are transmitted over lossy network, so we could find proper sending order to improve the quality as fast as possible, considering both the model property and network conditions. To achieve this objective, we have proposed an analytical model. The analytical model estimates both the expected value and the distribution of the decoding time of each node. We derive closed-form expressions in two extreme cases: the *ideal case* where all nodes are considered independent, and the *worst case* where a node depends on the all previously sent nodes. Any packetization algorithms will lead to a dependency structure between the packets that lies between these two extreme cases. The difference between the number of decodable packets for these two cases therefore provides insights to the importance of dependencies on the decoded model quality and gives us an indication of how much improvements we can get if we intelligently group the nodes into packets. The gap between the two extreme cases is significant only during a short time. So, the main observation from our analysis is that dependencies matters only during the first few seconds. After that, the dependencies among the packets will not affect the number of decodable packets. Further, in progressive models, the importance of the nodes usually decreases as the model becomes incrementally refined. It is true for vertex splits as the triangles size decreases. It is also true for plants because the size of children branche tends to be less that the size of parent branches. Thus, the contributions of the later nodes to the decoded quality of the model are less than the contribution of previous nodes. First, the fact that the streaming strategy does make a difference during the first instant of receiving the model also means that it does matter for interactive application. For example, when navigating withing a virtual world, a user going at high speed will only see objects for a short time. Second, this observation is good news - regardless of how large the progressive model is, only dependencies among nodes sent during the first few seconds matter. Thus, any packetization algorithm only needs to focus on the vertex splits sent during this initial period, reducing the computational costs significantly.

4.4.3 The *greedy* packetisation strategy

The analytical model was used to estimate the quality curve based on both the model property and network condition. We estimated the quality at time t as the sum of importance of decoded vertex splits. In progressive streaming, the quality of the model on the receiver increases over

time, and plotting this quality versus time gives us a quality curve. Because nodes contribute differently to the quality, the quality curve depends on the decoding time and decoding order of the nodes. So we evaluated and compared different streaming strategies of the same progressive mesh by comparing the quality curves. We show that the actual quality curves, and the expected quality curves give similar comparison results. Therefore, we can evaluate a sending order by predicting the expected quality curve with our analytical model given the network condition and the mesh property. To show that the insights from the theoretical model can be applied in practice, we extensively verified our model under different realistic conditions, using different progressive models, network conditions, and quality metrics. Extensive quantitative results are given in [Cheng 2011, Mondet 2009b].

Moreover, we used the predicted quality to design streaming strategies, particularly, packetization of nodes. Consider a node v. We need to decide whether we should pack v into the current packet. First, note that if there exists a parent of v that has not been packed, then we should not have packed v (if a parent of v arrives later than v, v cannot be decoded anyway). Thus, we only consider nodes whose parents have all been packed. Now, consider what would happened if we pack v into the current packet, versus the subsequent packet. We estimate the difference in quality v between the two cases, and call v the penalty. Minimizing the penalty maximizes the difference in decoded mesh quality. The *greedy algorithm* packs the node with highest penalty at each step.

4.4.4 Experiments

Here, we show results comparing the FIFO strategy based on importance of nodes, and the greedy strategy, maximizing the expected quality from our analytical model.

Progressive meshes
As mentioned, we have considered two quality metrics for progressive meshes, the split edge length and the Hasudorff distance. This last metric is neither additive, nor order-independent. However, the results shows that, in the first seconds, the Hausdorff distance is slightly higher than the predicted quality from our model. This difference is due to the change in the decoding order caused by packet losses. Since the decrease in Hausdorff distance is often larger when a vertex split is decoded earlier, the importance becomes higher than what we predicted. The two qualities, however, are still close to each other, indicating that our model is reasonably accurate even if we use a quality metric that is not additive and depends on the decoding order. The experimental results show that our model works well under realistic conditions despite the simplification and assumptions we made during modeling.

Comparison of FIFO and greedy strategy
Figure 4.21 compares the relative improvement in quality of the proposed greedy algorithm with respect to the FIFO algorithm on the Happy Buddha. We run one thousand different transmissions for each strategy. After some packet loss, Figure 4.22 shows that the two strategies can lead to significantly different qualities for small t.

Here also, we have measured difference in the model qualities between FIFO and greedy strategies over lossy network conditions. Figure 4.23 shows the traces of sending four walnut trees from Singapore to France. Figure 4.24 show one of these tree recovering from a depency accident; a

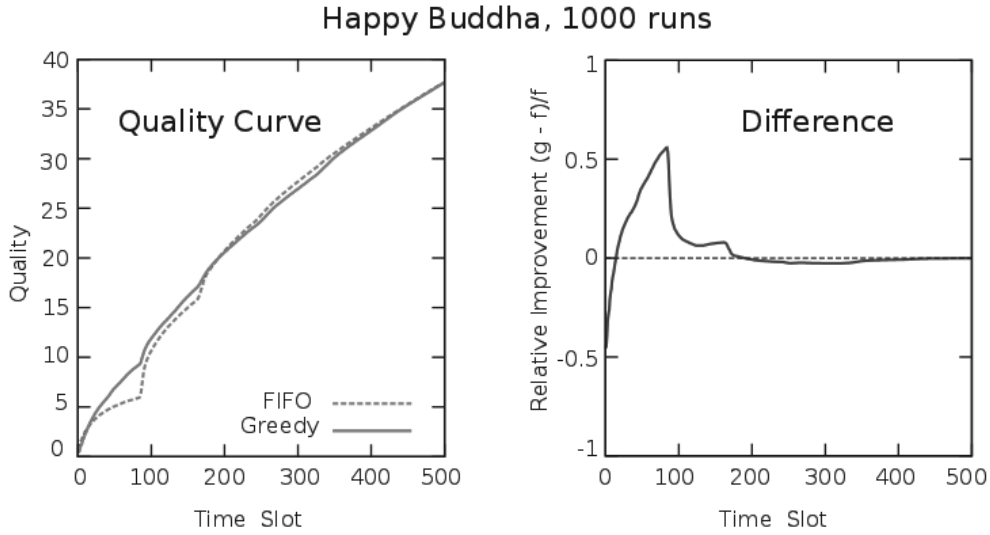


Figure 4.21: Quality curve for 1000 transmissions of the Happy Buddha model using FIFO and Greedy.

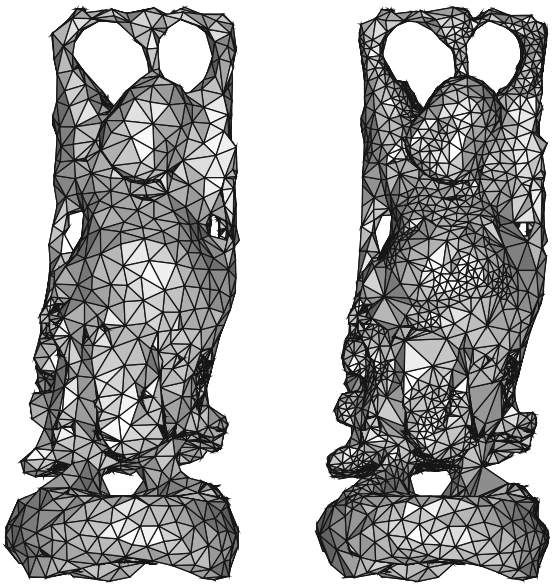


Figure 4.22: The rendered *happy Buddha* Happy Buddha for t small. Left: FIFO with quality 5.96. Right: greedy with quality 9.29.

early node has been lost, and the greedy strategy treatment of the retransmission policy performs better.

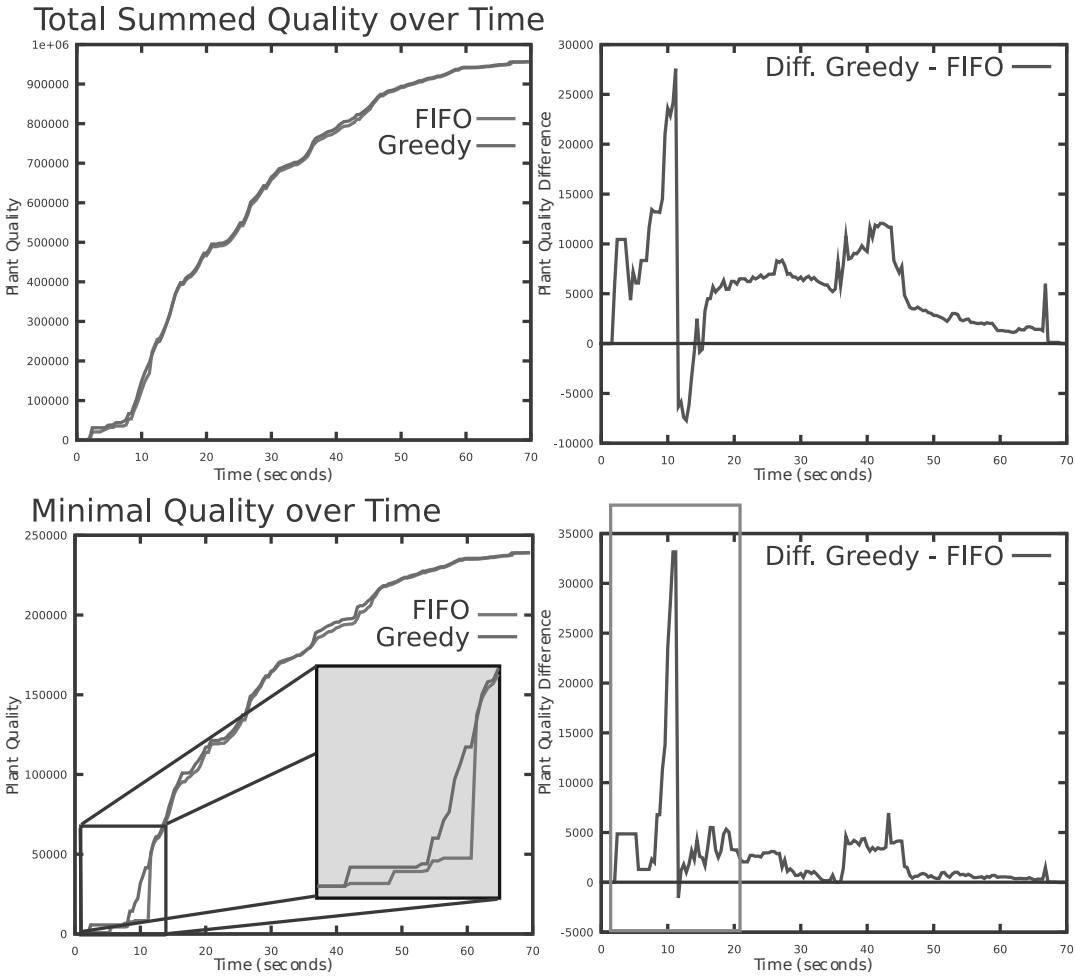


Figure 4.23: Quality curve for one transmission of four Walnut tree models using FIFO and Greedy.

4.4.5 Conclusion, limitations and perspectives

We first conclude by reflecting on what we learn from our model. The main insight is that if each lost packet is immediately retransmitted upon the packet loss detection, then the dependency only matters in the first several round trip times. Therefore, the effect of dependency is only significant in the applications requiring high interactivity. The most important insight we gain is that the effect of dependencies among vertex splits on decoded mesh quality when transmitted over a lossy network is limited to the first few Tds. This has several implications. Since Td is typically small, the effect of dependencies is not going to matter for non-interactive applications. Thus, we believe that existing research to reduce dependencies (for instance, mesh segmentation [65, 84] and packetization [35]) is not relevant in this context if retransmission is used.
For interactive applications, the decoded mesh quality in the first few Tds is crucial. We showed that FIFO ordering gives good enough average quality - even though a more intelligent greedy packetization can give better average quality. Besides packetization, our insight that only first

Figure 4.24: Example showing the structure of the same tree of the scene better ordered by the Greedy strategy.

Tds matters can lead to other transmission schemes. For example, the sender can protect the vertex splits sent during the first few Tds using enough FEC coding, ensuring that there is no losses; or the sender can increase the sending rate temporarily during this brief period to improve the initial quality. Our model is still useful here. For instance, sending FEC would decrease the loss probability of some packets but would delay transmission of new data. Temporarily increasing the sending rate may also increase the loss rate due to congestion. Our model can characterize this trade-off and thus can guide the sender in deciding whether to send a FEC packet or new data. Our formula for expected decodable quality can similarly guide the sender in deciding how fast it should send during the first Td period. To show that the insights from the theoretical model can be applied in practice, we extensively verified our model under different conditions, using different progressive meshes, network conditions, and quality metrics. The experimental results show that our model works well under realistic conditions despite the simplification and assumptions we made during modeling.

4.5 3D preview streaming

In this section, we briefly present follow up work on 3D mesh streaming. This first application considers *3D preview streaming*. For that, we shall see that the quality metric needs to be considered dynamic; that is, the importance of the nodes will depends on a (changing) viewpoint (section 4.5.2). Then, we define the camera path from keyviews (section 4.5.3). Finally, we propose different solutions to adapt the preview depending on the avialability of the bandwidth (section 4.5.4). Then, quantitative evaluation of the different solutions as well as qualitative results (involving user's feedback) are presented (4.5.5).

4.5.1 Motivation and Definition

Many platforms propose nowdays to visualize 3D models using *WebGL* or also propose models to download for populating 3D virtual worlds[9]. Usually, to preview a 3D model (e.g. to choose to visualize it in 3D), the content provider shows one of several images/thumbnails or may play a short video. The idea of *3D preview streaming* is to propose a preview of the 3D model *while* dowloading. Like in video-based previewing, we display the 3D model at the client viewed from a

[9]see for example `http://www.3d-conform.eu`

moving camera that navigates along a predefined camera path. Unlike video-based previewing, *3D preview streaming* downloads and renders part of the 3D model visible from the current viewpoint.

4.5.2 Dynamic quality metric: adaptation to the viewpoint

The first step is to adapt the streaming of a progressive mesh to favor the part of the model most relevant to the current view point of the client. In order to do that, the quality metric is considered dynamic, that is, vertex split importance is dependent of the viewpoint.
First, vertices that do not visually contribute to quality improvement are not considered at this point for streaming. Note that applying frustum (the one ring of the vertex is outside the frustum) and backface culling (normals of the faces of the one ring all point backwards) to a vertex in not sufficient. One also needs to consider if any of the descendant vertices (in the DAG dependency structure) contributes. For the vertices that may contribute, the chosen metric is the **screen-space area** of the one ring of the vertex. This metric can be computed on the GPU. Moreover, this metric can provide a way to do filtering: that is, if the object is far away or the client has a low resolution screen, we can avoid the sending of all vertex splits below a threshold. For example, we avoid sending the split of a vertex whose screen-space area fall below one pixel.
3D mesh preview streaming is a specific form of view-dependent progressive mesh streaming. In 3D mesh preview streaming, there is no interaction between the user and the 3D model. The sequence of views (viewpoints and synthetic camera parameters) is predetermined in the form of a parametric camera path $P(.)$, which is known by both the server and the client.

4.5.3 Bandwidth-aware camera path

We now present our method to compute the camera path. We start from a set of representative views, called *keyviews*, is given as input. These keyviews can be set manually by the content provider, or can be automatically determined using the existing methods ([Kamada 1988] chose non-degenerated views; some more recent work maximizes a criterion based on the 3D geometry: the number of polygons seen [Plemenos 1996], the total curvature [Sokolov 2005], the viewpoint entropy [Vázquez 2001], or the view stability [Vázquez 2009]; others use perceptual criteria gathering user experience (e.g. [Burelli 2010, Picardi 2011]; finally, [Dutagaci 2010] give a nice comparison of state of the art methods).
A good camera path needs pass through the keyviews, and travel directly and smoothly between two neighboring keyviews on the path. For preview streaming, we add two constraints: (i) the amount of data shared between two successive keyviews should be maximum, in order to reduce the fluctuations in bandwidth requirements as the viewpoint moves along the camera path, and (ii) the camera path must be smooth with no sudden changes in the gradient of the path, especially near the keyviews.
We construct a weighted complete graph $G = (V, E, w)$ where each vertex in the graph is a keyview. Every two keyviews are connected with an edge (u, v) whose cost $w(u, v) = -Sp(u, v)$, the size of vertices visible from both keyviews u and v. The cost function favors visiting a neighboring keyview sharing a maximum of visible triangles. The problem of finding the sequence of keyviews to visit that would minimize the total cost is then equivalent to the traveling salesman problem, which is NP-complete. For our use case, the number of keyviews is small (we use 12 in our evaluation) and the path is computed offline, so we used a brute force approach. From the ordered sequence of keyviews, we compute the parametric camera path P as an interpolating

Catmull-Rom splines, since such curves are C 1 -continuous. We chose the chordal parameterization on the position of the viewpoints proposed by Yuksel et al. [C. Yuksel 2011]. There are three advantages in choosing this parameterization: first, the behavior does not depend on the relative difference in positions of the keyviews. In particular, the keyviews are far from being uniformly distributed (since they are chosen for their visual interest), the chosen interpolating curve does stay close to the line joining only successive nearby positions. Second, between two far away positions corresponding to two successive keyviews, a relatively long path is derived, leading to intermediate keyviews that stay at a relatively constant distance to the object. The chosen model quite naturally avoids collision with the model. Finally, the parameterization is fairly regular, which is important for us as we sample views along the parametric path: regular parameter thus lead to quite regularly sampled positions on the path. We use this property in the following to get an approximate arc-length parameterization.

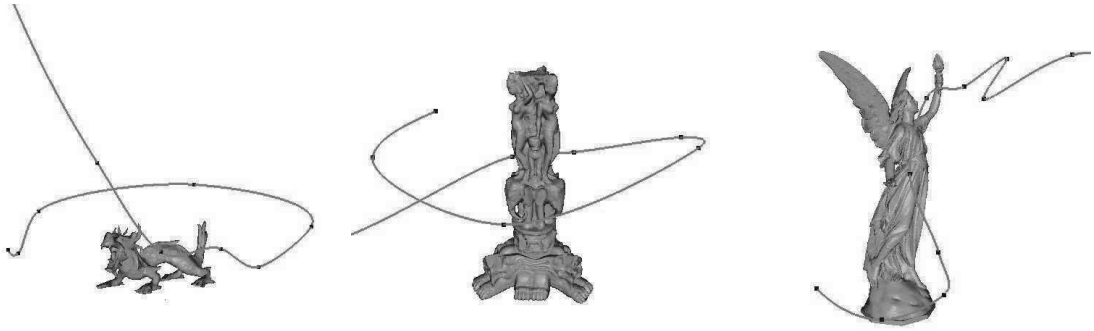


Figure 4.25: Camera Path of Asian Dragon, Thai Statue, and Lucy. The black points are the camera positions of keyviews (we picked 12 keyvoews manually). The 3D models come from the Stanford 3D Scanning Repository.

4.5.4 Adapting to bandwidth variation

We discretize the camera path into N sample viewpoints corresponding to parameters taken at an equal distance d. The time taken for the camera to travel between two sampled views is 1 unit time. The camera speed is therefore d. We assume the camera speed is constant in the ideal streaming algorithm (unlimited bandwidth). The server computes $M(t)$, the set of vertices visible from any point along the curve between $P(t)$ and $P(t+1)$ (inclusive) that has not appeared before in any set $M(t)$ for $t < t$. These are the set of vertices that has not been seen before if the client follows the camera path from $P(0)$ to $P(t)$. The size of the data in $M(t)$ can also be precomputed. We denote the size of $M(t)$ as $B(t)$ (in bits). The playback rate, the rate in which the data are consumed and display, is $B(t)$ (in bits per unit time) in this ideal case where there is enough bandwidth. The server algorithm is fairly simple: for each time $t = 0, 1, ..., N2$, the server sends $M(t)$ with the data rate of $B(t)$. Suppose we start the clock $t = 0$ at the client one unit time later (plus some delay to account for network latency and jitter) than the server. Then, assuming that the data rate of $B(t)$ is sustainable, this algorithm ensures that the client will receive all data of $M(t)$ by time t for rendering within the next unit time.

Now we consider the case where the server has a maximum sending rate of r. If at some time t where $B(t) > r$, the client would not have received enough data to display the mesh at full resolution. To counter the mismatch between the playback rate and the sending rate, we propose

four strategies:

- STOP-AND-WAIT: the camera stops to wait for the additionnal data to arrive (we additionally propose to rather stop at a keyview since it offers an interesting viewpoint);
- REDUCED-QUALITY: the camera keeps the initial speed are renders the avialable 3D content;
- REDUCED-SPEED: the camera slows down giving time for 3D data to arrive at the client;
- KEYVIEW-AWARE: this last solution is a compromise between REDUCED-QUALITY and reduced-speed taking into account the importance of the viewpoint; keyviews are considered priviledge viewpoint to slow down, and the quality should rather be reduced at less important viewpoints.

4.5.5 Results

In this section, we present the evaluation results of the three last 3D preview streaming schemes proposed in this paper, as well as a user study to measure the perception of different solutions.

4.5.5.1 Experimental setup

We use the 3D models of Figure 4.25. Because Lucy is the largest mesh used, we evaluate these schemes with Lucy. We render the model on a 500×500 pixels canvas with OpenGL. The progressive mesh of Lucy has 264 vertices in the base mesh and 14,027,755 vertex-splits (each requires 32 bytes). Note that it is larger than a non-progressive mesh because we need to support both view-dependent streaming and out-of-core rendering of the model.
For evaluation, we implemented five schemes in 3D preview streaming: GROUND-TRUTH, STOP-AND-WAIT keyview, REDUCE-QUALITY, REDUCE-SPEED, KEYVIEW-AWARE schemes. GROUND-TRUTH assumes the outgoing bandwidth is unlimited and therefore the preview always has the best quality and constant camera speed.
The data rate $r(t)$ starts from 100 KBps and switches between 25 KBps and 100 KBps every 30 seconds. We uses this range of values as it roughly matches the variations of bandwidth required to transmit the 3D model we used.
Table 4.3 shows the duration of the preview and the amount of received data for the different schemes. The duration indicates the amount of time the users would need to watch through the whole preview. REDUCE-SPEED leads to preview that lasts more than twice the duration of the GROUND-TRUTH preview, while REDUCE-QUALITY leads to three times less vertices being received, reducing the mesh quality. KEYVIEW-AWARE has duration and quality that falls in between both schemes. Figure 4.26 illustrates the quality of two selected keyviews when streamed using the three non-ideal schemes.

4.5.5.2 Quantitative results

The plots of Figure 4.27 illustrate how the speed of the camera and the size of downloaded data of REDUCE-SPEED, REDUCE-QUALITY, KEYVIEW-AWARE change over time. Since the schemes lead to different camera speed, the y-axes of the graphs are shown in units of the path length. The vertical lines in the figures indicate the position of the keyviews.
We first focus on the REDUCE-SPEED scheme (in red). Initially, since there are significantly more data to transmit the camera speed drops significantly compared to other schemes. The quality

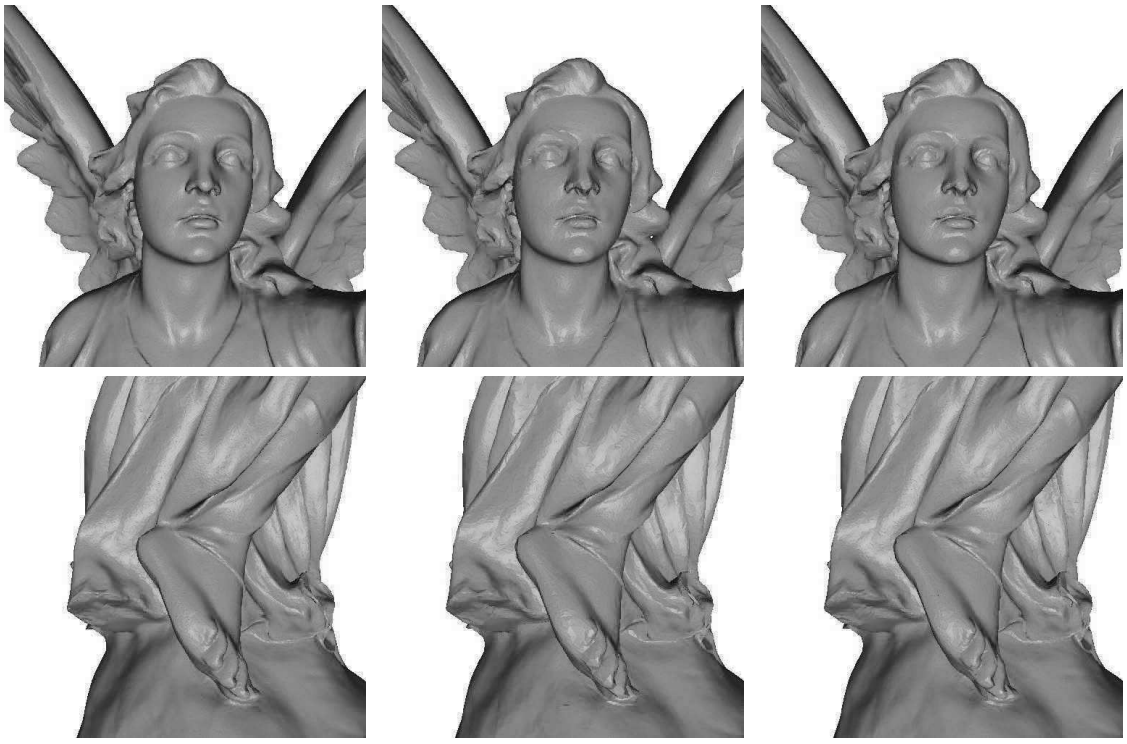


Figure 4.26: Mesh quality of selected views: left: REDUCE-SPEED, middle: REDUCE-QUALITY, right: KEYVIEW-AWARE.

Scheme	Duration	Received Vertex-Splits
GROUND-TRUTH	84 sec	14,989 KB
STOP-AND-WAIT naive	202 sec	14,989 KB
STOP-AND-WAIT at-keyview	210 sec	14,989 KB
REDUCE-QUALITY	88 sec	4,986 KB
REDUCE-SPEED	219 sec	14,989 KB
KEYVIEW-AWARE	148 sec	10,645 KB

Table 4.3: Statistics of schemes.

of REDUCE-SPEED, however, is the highest. Next, we illustrate the REDUCE-QUALITY scheme (in blue). The camera speed is constant, but the quality is the lowest, and therefore fewest amount of data being transmitted.

The graph for the KEYVIEW-AWARE scheme (in green) is the most interesting. The camera speed fluctuates between S_{mid} and S_{max}, during a period of high sending rate. After 30 seconds, the camera speed is forced to slow down due to low sending rate. The inter-dependency between the camera speed and data rate causes both values to fluctuate during this period. The number of polygons rendered stays between the other two schemes, but peaks at keyviews. This observation is more obvious at length 3700. There is an increased in the number of polygons rendered compared to REDUCE-QUALITY scheme.

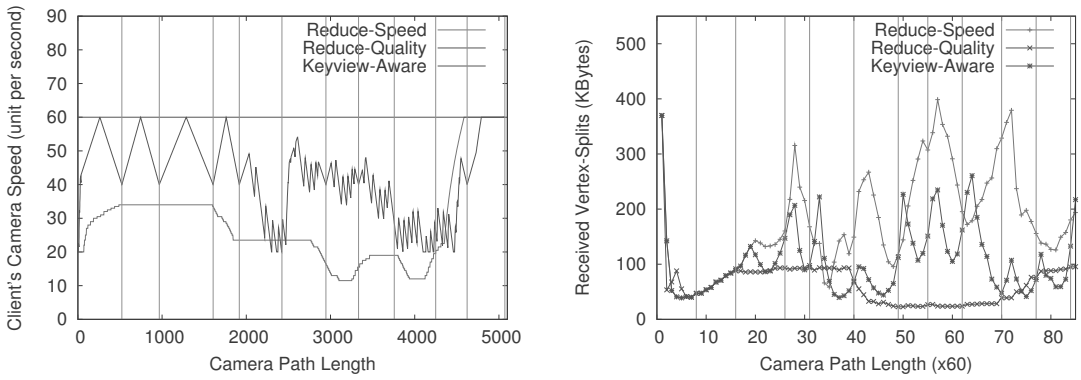


Figure 4.27: Evolution of different schemes in our experiments.

4.5.5.3 User study

A user study has been performed to get qualitative evaluation of the proposed strategies. The protocol and set-up, as well as more complete evaluation results can be found in [Zhao 2013]. Overall, the user study rules out both the STOP-AND-WAIT and the REDUCE-SPEED because the preview is too long (less than 10% found the speed of these solutions acceptable). Whereas users are ready to compromise on the quality (60% found the quality of REDUCE-QUALITY acceptable and 96% found the quality of KEYVIEW-AWARE acceptable), they significantly prefer the KEYVIEW-AWARE solution (best ranked among all strategies) offering a compromise between (a smooth) speed and quality.

4.5.6 Conclusion, limitations and perspectives

We have proposed in this work a first step on previewing while downloading, using 3D preview streaming. There are many direct extension to this work, like finding the keyviews automatically, or even more ambitiously allow some user interactions.
Taking a step back, the question is what is the best way to preview a model. Here, we have also started to consider users feedback, and probably, the best way will depend not only on the system parameters (e.g. the available bandwidth, or the client's device) but also on the user himself. Providing adaptive user content is necessary, and probably a next challenge is to adapt it to the technical context but also to the usage.

4.6 Streaming 3D to mobile devices

4.6.1 A first step

In a first step we have developped a mobile application to stream 3D to a mobile platform [Doran 2009]. This demonstration used the model presented in section 4.3 (with leaves as shown in 4.3.6.1).
Two facts emerged from this work: first, 3D models can be large, but they can also be **small**. Indeed, because we work with a rather simple phone (in 2009!) the limited resolution of the screen led to discard small branches that did not appear due to the lack of resolution. The remaining size of the 3D model, with our reprensentation, was only 20 Kb, which appears to be smaller than most images. Second, rendering on a mobile phone was a challening task (in 2009!) in terms of complexity for the phone, but also in terms of ressources. We would be tempted

to just let time pass by to get more computing power, on CPUs but also on GPUs (letting the classical Moore's law work for us), but this approach somehow does not work (anymore): portable and mobile devices, working on more limited ressources and lighter communication channels, want the same services than users on PCs. Services means accessing the same content, in our context 3D content.
That led us to the next idea, if powerfull clients/users (that is, with large ressources: good communication channels and powerfull CPUs and GPUs) are going to access the same content than light clients (with limited ressources), could light clients benefit from the work of more powerfull ones? The next section will sum up our first approach in this direction: easing transmission of 3D content to light clients, but also simplying their rendering task, uding the support of more powerfull clients. The next section summarizes our work in this direction.

4.6.2 3D adaptation for Transmission and Rendering

4.6.2.1 Motivation and definition

Virtual 3D environments can be accessed by concurrent users, sharing 3D content or space. Second life, or more recently Kitely[10] or Cloud party are softwares offering navigation in 3D worlds. Last generation mobile devices are indeed powerful enough they can still only access a simplified version of NVEs (e.g. using Lumiya[11], a NVE client for Android phones). The goal of the proposed work, that we call *Pear Assisted Redering* is to benefit from the heterogeneity of users accessing these virtual worlds. For second life 10^{th} anniversary (June 2013), Linden labs reports "*36 million accounts were created over the last 10 years. Today, more than 1 million people visit Second Life monthly. Games, events and Adventure/fantasy dominate the Destination guide.*" We want to benefit from this concurrent access, and in particular from the heterogeneity of users for easing the navigation of light client into the NVE (network virtual environment). First, light client usually have limited communication channels, so streaming the 3D objects may be costly in terms of bandwidth. Second, light client also often have reduced rendering capabilities compared to regular computers and GPUs on cell phone increase the power consumption drastically [Mochocki 2006]. A common approach to reduce the rendering workload at the mobile client consists in migrating the rendering tasks to a server (e.g. [Noimark 2003, Cheng 2004, Shi 2009]) or to a Cloud [Shi 2010]. However, this approach is not always scalable: Sterkin reports a maximum of 14 users on a high end server [Sterkin 2008]. Our proposal is to ease the rendering task of a light client, and reduce the size of the transmitted 3D content, by recycling rendered views of nearby, more powerful clients. We say that the mobile client is assisted by an *assistant* that renders on behalf of the mobile client, and we call the mobile client the *assistee*.
These drawbacks of the existing proposals motivate the need for a scalable method to support rendering of dynamic scenes from a complex multi-user virtual environments on a mobile device. This section introduces a new technique to reduce the rendering workload of an assistee, by distributing the rendering tasks to other clients, or peers, in the same virtual world. We call this approach peer-assisted rendering. Clients accessing the virtual world are heterogeneous in nature. Some clients are running on hosts with good rendering capabilities (most desktops are equipped with GPU). These clients can act as assistants, render parts of the scene separately, and send them to the assistee. We call the partial scene rendered at the assistants rendering elements. The assistee then merges the rendering elements to create its view of the scene.

[10] `www.kitely.com`
[11] `http://www.lumiyaviewer.com`

4.6.2.2 A feasible solution

The goal of this study is to prove the concept for peer-assisted rendering. We consider the scheme where a mobile device prof- its from rendering work already done, and cached, by other peers. The assistee will have to build its view of the scene by using image warping from the rendering results (object images) from other viewpoints (c.f. Section 3). Therefore, the assistee needs to find other peers that can provide him with, at least, partial rendering results corresponding to his viewpoint, i.e., other peers that see a part of what he sees. In order to show that this peer-assisted scheme is viable in the current usage of Second Life, we need to define notions of similarity for avatar views. In the remaining of this section, we build two similarity criteria which reflect how much the assistant may help with object images or for background warping.

Quantifying View Similarity and Choosing Assistants

We define similarity metrics as a measure between the assistee and a set of candidate assistants. These metrics measure the capacity of a set of potential assistants in providing the objects impostors and background to the assistee. We define two similarity measures, *object similarity* and *viewpoint similarity*. The *object similarity* counts the ratio of the number of objects that falls into the viewing frustum of both the assistant and the assistee, to the number of objects in the viewing frustum of the assistee. The idea here is that, if an object falls into the frustum of the assistee, it is needed by the assistee, and if it also falls into the viewing frustum of the assistant, then it can be pre-rendered by the assistant. There is an additional factor to consider, however. The rendered object by the assistant would not be helpful to the assistee if the difference in the viewing angle of the assistant and assistee to the object is large. In the worst case, the assistant and assistee could be facing each other with the object in between. In this case, the impostor from the assistant would be useless. As such, we filter out assistants whose differences in viewing angle with respect to the assistee's viewing angle is larger than a threshold, and set the similarity to zero. This metric naturally generalizes to more than one candidate assistant, by considering the number of object shared between the assistee and any of the candidate assistants, thus taking into account the complementarity of the assistants. The *object similarity* is indicative of the amount of data that can be shared between the assistant and the assistee, but it involves point in polygon query, which could be expensive, especially on the assistee that is already resource-constraint. We introduce another similarity measure that is easier to calculate and approximate the object similarity, called viewpoint similarity. *Viewpoint similarity* computes the area of the intersection between the view frustum of the assistant and the assistee. To compute viewpoint similarity, we only consider 2D view frustum in the xy-plane. We ignore the z-angle, the angle between the vector z_camera and the horizontal plane (c.f. Figure 4.28), since most users have a horizontal viewing direction. We also ignore the near-plane. Hence, the 2D viewpoint of a peer is modeled by a triangle: the position is the vertex, facing the edge projection of the far-plane, and the two remaining edges are defined by the sides of the 3D frustum (see Figure 4.29).
Two viewpoints are then compared using the overlapping area between two 2D viewpoints, that is, the projection of the intersection volume on the horizontal plane (c.f. Figure 4.28). We compute the similarity as the ratio between the overlapping area and the area of the assistee. The similarity for a pair of avatars thus ranges between 0 and 1, with 1 representing the same viewpoint (c.f. Figure 4.29). As in object similarity, we set the similarity between two avatars to zero when the angle between their view directions is larger than a threshold (we use the default horizontal FOV setting in Second Life, which is 91 degree). To extend this metric to multiple assistants, we compute the intersection between the assistee's 2D viewpoint and the union of all assistants' 2D viewpoints whose viewing angle lies within the threshold. In [Zhu 2011] (section

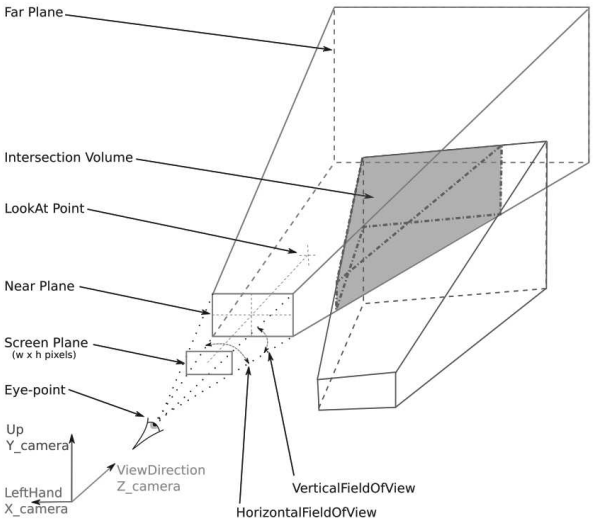


Figure 4.28: The viewing frustum (in red), and its intersection with another frustum (in blue).

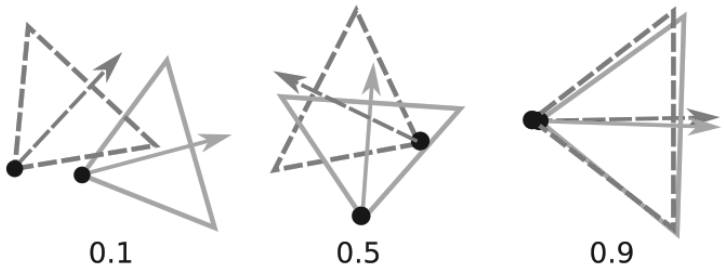


Figure 4.29: Pairs of 2D viewpoints and their overlapping area ratios. Viewpoint similariry is computed by filtering with the angle between viewing directions.

6), we show that using this two measures gives very similar results, and that using only the viewpoint similarity measure is sufficient. As a consequence, the results presented here will be in terms of the viewpoint similarity only.
To select the best assistant with a reasonable computational cost, we use a greedy algorithm: an assistee chooses iteratively its assistants to maximize, at each step, the global similarity with the set of assistants.

Analyzing the disponibility of assistants
We focus on answering another fundamental question. Will there be enough supply of assistants with common viewpoints? To answer this question, we analyze a large collection avatar traces from Second Life to study the viewpoint similarity between the avatars from a real, popular virtual world.
We use traces from three regions in Second Life: Sunland, Japan Resort, and Freebies. We choose a one-hour period to analyze the position and viewing parameters (an *event*) of each avatar every 10 seconds. Previous analysis of traces in Second Life have shown little hour-to-hour variability in the characteristics of the avatar mobility. The number of avatars/events recorded are 71/5786; 61/5912; 53/2516 in Freebies; Japan resort and Sunland respectively. For each region, for each time t in the trace, and for each avatar a that appears in time t in the trace, we do the following. We compute the viewpoint similarity between this avatar a and find the avatar with the highest viewpoint similarity with a. In other words, we find the highest viewpoint similarity over all possible pairs (a, i) where $i \neq a$. By averaging this value over all time t where a appears in the region during the period of analysis to obtain the average maximum viewpoint similarity $VS_{max}(a)$ for each avatar a. We plot the CDF for $VS_{max}(a)$ for the three regions in Figure 4.30. In the same figure, we also plotted the CDF for average maximum viewpoint similarity for each avatar a and two other avatars, and similarly, for three other avatars.

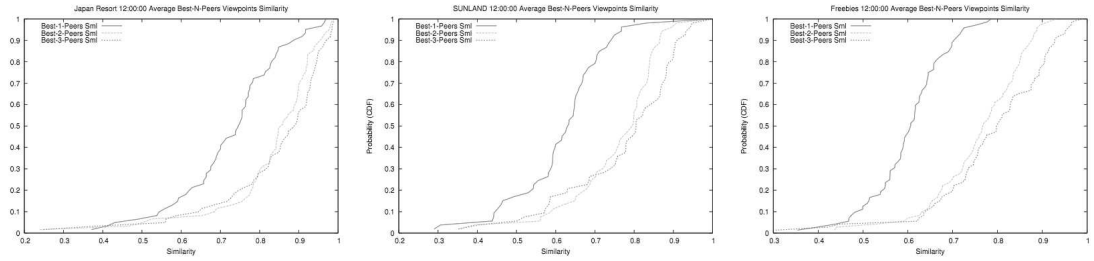


Figure 4.30: CDF for Viewpoint Similarity for three regions: Japan Resort (left), SUNLAND (center), Freebies (right).

The results can be interpreted as follows. Take Japan Resort for example. Consider an assistee that uses k assistants. If k is 1, then on average, more than 20% of the avatars have an assistant that has viewpoint similarity of about 0.7. If two assistants are used ($k = 2$), then every avatar can find two assistants with combined viewpoint similarity of 0.85 or above. Following this interpretation, for all three regions that we analyzed, we can see that there is a significant amount of avatars with high viewpoint similarity, especially if we use two assistants. In all cases, more than half of the avatars can find two assistants with viewpoint similarity above 0.6. Figure 4.30 also indicates that, as the number of assistants chosen increases to three, the improvement in the viewpoint similarity is limited, and in some cases, even reduces. The decrease is because for

some assistees only two candidate assistants are passing the filtering on viewing angle difference: the number of samples for best-2 and best-3 is thus different.
These trace analysis shows that in a practical NVE, even when there are only tens of avatars in a region, it is feasible to find enough peers that share similar viewpoint and a common set of objects in the viewing frustum, which is one of the fundamental conditions for peer-assisted rendering to be practical. This result further supports the efficacy of the peer-assisted rendering.

4.6.2.3 A first solution

Now that we have seen that there are enough potential assistant in popular areas of a NVE for a assistee, we propose a scenario for implementing peer-assisted rendering.

Image-based rendering: warping
To limit the amount of data transmitted to an assistee, we can use, instead of a rendering algorithm, the warping algorithm proposed by McMillan [McMillan 1995, McMillan 1997]. The idea is to generate an image of a 3D scene from a image of the same scene with a similar viewpoint. Exact warping can be done for certain camera moves: homographies are exact for rotations around the *eye-point*, or also, for arbitrary motion of a planar object. Image warping using the depth map is not exact, in the sense than exposure gaps can appear, when hidden surfaces for the assistant are visible for the assistee: information is missing.
However, the advantage is to transmit only images of partial rendered scenes, along with the depth maps. The warping approach has already been used in server-based solutions for easing the rendering on mobile clients [Chang 2002, Bao 2006]. Shi et al. [Shi 2009] use warping for mobile clients in a slightly different context: in their setting, the input data is a 3D video stream. The latency of the latter setup has been then improved by warping from multiple images [Shi 2010]. Another approach is based on pre-rendered, buffered panoramic images. In Boukerche and al. system [Boukerche 2007], the server receives the mobile user's viewpoint and warps from a panoramic view in cylindrical coordinates for the client. Lei et al. [Lei 2004] suggested that the client warps the scene himself from a portion of the panorama.

Decomposition of the scene
We choose to use warping; however, to avoid exposure gaps, we consider different objects of the scene in different *rendering elements*. A fundamental question that peer-assisted rendering approach needs to address is how to split the rendering tasks and what is a rendering element. In our design, we choose to have three types of rendering elements, to balance between the rendering load on the assistee and the assistants, to maintain interactivity, and to minimize visual artifacts. The first type of rendering elements consists in the furthest objects, the sky, and the ground. These objects are rendered as one single depth-enabled image by the assistants according to their own viewpoint. We call this image the *background impostor*. Since the assistants are rendering them for their own use in any case, rendering the background impostor does not incur much additional overhead. The second type of rendering elements consists in the static near objects. These objects are rendered as individual images by the assistants. We call these rendering elements the *object impostors*. The objects are also rendered according to the viewpoint of the assistants, so it incurs very little additional overhead for them. Moreover, the assistants may already use some of these rendering elements for themselves (e.g. for rendering shadows, and reflections). The impostors and the corresponding depth maps are sent to the assistee, which warps them to its view point. The final type of rendering elements consists of dynamic objects, such as other avatars. These objects are rendered locally by the assistee as

regular 3D objects.

Rendering at the assistee

To generate the rendered scene, the assistee combines the three classes of rendering elements, two of which are warped, and one is rendered regularly. The assistee, upon encountering a new object in its frustum or upon experiencing large visual artifacts in rendering of an object, sends a request to an assistant (based on the assistant selection algorithm) for the corresponding impostor. The assistant renders the requested objects as an impostor and sends them, together with the corresponding depth map, to the assistee. The same is done for the background impostor. Instead of making the assistants generate the impostors according to the assistee's viewpoint, we choose to use warping at the assistee. The assistee knows the viewpoint of the assistant from which the impostors come from, and uses this information to warp them to its own viewpoint. This approach improves the overall latency for two reasons. First, the assistant does not need to get updates of the assistee's viewpoint, which would increase the network round-trips. Second, by reusing the assistants' depth-enabled images, the assistee can move in the scene and continue warping without requiring any additional information. The scene at the assistee now consists of a warped background, warped impostors and moving objects within distance d from the eye-point (including the avatars). The assistee now renders the scene using regular graphics pipeline. The composition of the three types of rendering objects is also performed by the last step (Z- buffer) of the graphics pipeline, using the depth information of the 3D objects of the background and of the impostors. Figure 4.31 shows an example that illustrates the whole process and a ground truth rendered image for comparison.

Computational Complexity

The complexity of the classical rendering process depends on both the number of polygons in the scene and the number of pixels in the screen. The complexity of a warping pass, however, depends only on the number of pixels of the warped image. Thus, the traditional 3D graphics pipeline does not benefit much from the smaller display size of a mobile device, whereas warping does [McMillan 1995]. The complexity of warping is comparable to the complexity of the rasterization step of 3D rendering. Hence, when warping an object, the time spent in the perspective transformation step and the projection step is saved, compared to 3D rendering. Moreover, grouping the objects simplifies warping further: in 3D rendering, objects are rasterized independently, whereas objects being warped together lead to a single pass on the pixel array. In particular, warping the background objects into a single image avoids considering the objects that are far away one-by-one. Note the resolution of the grouping (how many objects per impostors) may be chosen: more object per impostors leads to less impostors, which lowers the complexity of the rendering at the expense of more visual artifacts (exposure gaps).

Results

We present here the rendering results at the assistee. To understand the effect of the number of assistants and to validate the viewpoint similarity criteria between the assistants and assistee, we show the rendering results when different number of assistants are chosen and when a different similarity threshold is used. Figure 4.32 shows the rendering results of the first frame if the assis- tant selection algorithm chooses 1, 3, and 5 assistants respectively. The ground truth is also shown for comparison. As expected, as the number of assistants chosen increases, the amount of visual artifacts reduces. Interestingly, with one assistant only, the majority of the

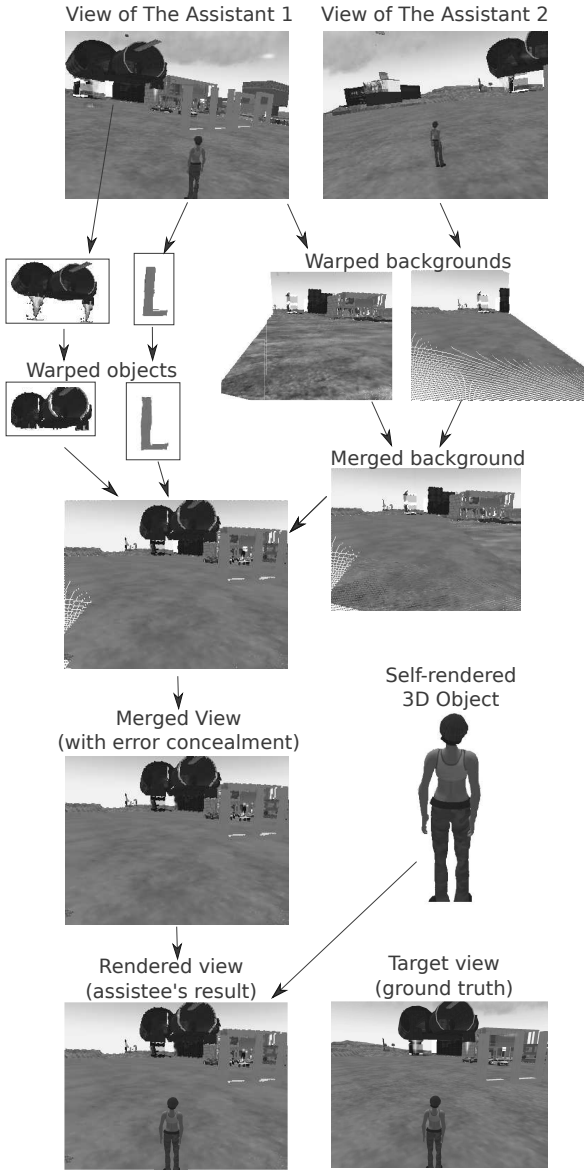


Figure 4.31: An example of hybrid rendering at the assistee: Assistant 1 provides the impostors and a part of the background, Assistant 2 provides a complementary part of the background, and the dynamic object (the avatar) is rendered locally.

Figure 4.32: Rendering results with different number of assistants and the ground truth. From Left to Right: one assistant, three assistants, five assistants, ground truth.

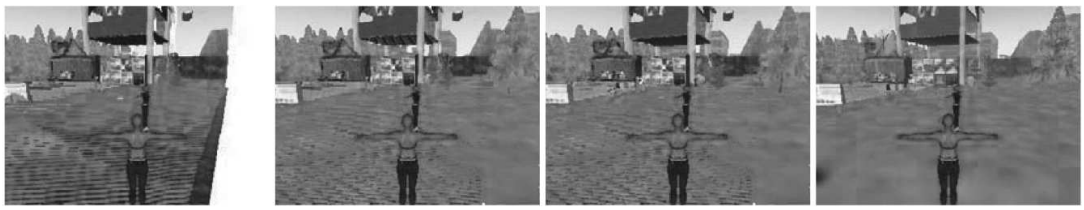


Figure 4.33: Rendering results with different viewpoint similarities and the ground truth. From Left to Right: 0.6, 0.8, 1.0, ground truth.

impostors and a large part of the background are rendered properly already. This observation supports the usefulness of viewpoint similarity and our assistant selection algorithm. Figure 4.33 shows the rendering results if the assistant selection algorithm stops running when the viewpoint similarity threshold exceeds a given value (0.6, 0.8 and 1.0). A threshold of 1.0 means that the assistant selection algorithm stops only after it fails to find any assistant that can improve the similarity. Here, we can visually quantify the impact of viewpoint similarity, even for impostors. As viewpoint similarity increases, the amount of visual artifacts reduces. Even with a viewpoint similarity of 1.0, however, we still have visual artifacts due to warping. Three kind of visual artifacts are visible: sampling issues (e.g., on the ground), exposure gaps (corresponding to white pixels on the image), and synchronisation issues (e.g., on the TV wall). Whereas exposure gaps issues are inherent to warping, we see here that using multiple assistants reduce significantly these issues.

Figure 4.34 the result of our peer-assisted rendering on this set of 10 assistants. Our assistee navigates through the scene as a user controls its trajectory. In this sequence, we are able to get an average similarity of 0.9. The number of assistants used varies between 3 and 6 (average is 4.1) depending on the position of the assistee. Figure 4.34 shows the rendered results for every 20 frames. The exposure gaps appearing in the sequence are due to the quantization of the rasterization, and could be solved by a more careful (and a little more complex) warping. Note however that no exposure gap due to occlusion between objects occurs, as in the classical warp from a single image.

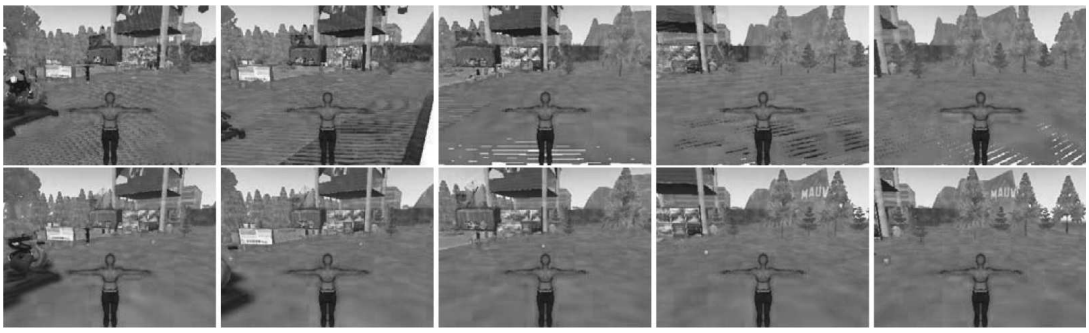


Figure 4.34: Every 40 frames of an image sequence generated by hybrid rendering: background, and nearby objects are warped, both avatars are rendered locally. The second row shows the corresponding ground truth images.

4.6.3 Conclusion, limitations and perspectives

Conclusion

There are many hurdles and research challenges in realizing peer-assisted rendering practically. This work, as the first introducing this concept, aims to establish a solid case for the efficacy of peer-assisted rendering. We chose to address two fundamental questions here: (i) are there enough assistants with similar viewpoints and common objects to help an assistee? (ii) what is the quality of the scene rendered at the assistee as user navigates around the scene? To answer the first question, we analyze mobility and object traces from a popular NVE named Second Life. We show that there are surprisingly many avatars with significant overlaps in viewing frustum, and most assistees can find enough assistants to cover a large number of objects in their viewing frustum. To answer the second question, we build a renderer that simulates the peer-assisted rendering process, and show that the resulting rendered quality is close to that if rendered fully by the assistee.

Limitations

The solution we propose here is a first step towards tackling the peer-assisted rendering framework, and, as such, does have many limitation. The first limitation is on the quality of the rendered scene. Even in the results of Figure 4.34, where we considered as many assistant as possible, the quality is still perfectible. Indeed, many papers have addressed this resolution problem in the context using splatting, meshing [McMillan 1997] or more sophisticated modeling like *Layered Depth Images* (LDI) trees (e.g., [Shade 1998]). We could to take advantage of our object-based implementation for solving the sampling issue.
Another issue can be seen on the TV screens (Figures 4.32, 4.33, and 4.34) where images from different assistants are mixed, causing incoherent mixed images. This problem highlights the synchronisation issue for dynamic object, here a TV screen, and could be fixed if we favored the most recent impostor.
Beside the modeling issues, and concurrently, the system issues need to be considered. That is, communication bottlenecks of the proposed solution should also be indentified, and both the grouping and resolution of the scheme adapted to meet some interaction constraints, like a given frame rate.

Future work

Besides the incremental improvement of the presented work, the question of navigating through, and interacting with, a large NVE on a mobile phone with limited computational capacity is challenging. The computational demand to render the 3D scene in NVE generally leads to low rendering frame rate, and reduces the interactivity. A common technique to improve the frame rate is to reduce the details of 3D content within the NVE. Previous work has proposed to simplify the geometry of 3D models [Jie Feng 2004] or filter out objects (by making them invisible) on the mobile client [Lluch 2005]. Despite improving the frame rate, these techniques reduce the quality of the rendered scene at some possibly relevant parts.

Our working direction for this problem is to keep requesting and warping pre-rendered object to ease the rendering task of light users. However, instead of requesting impostors like in the previous scheme, we propose that clients with more resources could fill in a data structure holding partial rendering of the scene. This scheme takes idea from the panorama approaches [Boukerche 2007, Lei 2004] and from the LDI (Layered Depth Images) [Shade 1998]. However, unlike in the panorama, the pre-rendered image are being updated, so the scheme may support

dynamicity. Moreover, because the content is fed in by users, the more popular or interesting part can benefit from more input data. We can imagine to refine te resolution of these relevant area. Also, no useless pre-rendering is done. In that case, assistee will be supported only (like in our previously proposed scheme) in popular areas. LDI tree could be a solution, but we have tried it, and it does not offer a reasonable computational complexity for offering interactivity. LDI tree works at a pixel (plus depth) level, with multiple resolution. Our previous scheme works with impostor containing one or more objects. So, a natural idea is to propose an intermediate representation to store pre-rendered images.

Perspectives
To conclude the section on mobile phone, a significant idea that needs to be taken into account nowadays by us, computer scientist, is that Moore's law does not really apply anymore. As computers are becoming increasingly powerful, and GPU offer alternative computing power, new, small, light and more limited devices are claiming for services and content similar to the ones available for real computers. Moreover, above being limited by resources and connectivity, these new devices also need to work efficiently in terms of energy. So, as there is need for more precise, and nicer models, that can be handled by powerful graphics cards, they is also a parallel need for adaptable, light and practical 3D models to be accessible by lighter clients.

4.7 Conclusion on 3D streaming, and perspectives

This section on 3D streaming has highlighted the need for streaming strategies adapted to 3D content, and in particular to the dependencies among the data chunks modeling a progressive 3D object. We have proposed solutions robust to data losses for streaming 3D content. Further work have used this scheme in a previewing setup. Except for the Pear Assisted Rendering part (section 4.6.2), we only considered geometric models. A natural extension is to consider model with (color) attributes, or also textured models. Of course, we can still benefit from the proposed 3D streaming schemes: for a mesh with attributes, we can imagine attaching the attribute(s) to the vertex and adapting the importance metric to take into account both the geometric and the attribute information at the vertex. For textures, streaming multiresolution images already follows some standards. However, the issue of interleaving the two streams (3D and images of textures) has to be considered. This problem has indeed already been considered in the context (and even more general context, since there is a temporal dimension) of free viewpoint video: the video encoding is interleaved with a 3D object model. For example, in [Theobalt 2004], an augmented reality scenario where the texture of the moving body is changed (and therefore coded separately) is proposed. So, a step further would be to consider streaming a colored or textured model.
If we do consider both the geometry and the textures modeling a 3D object, or a 3D virtual world, in order to propose a progressive streamable model, the classical texture vs. geometry compromise arises. Many representations have simplified the geometry but replacing it by texture. We already mentioned billboards for plants modeling; billboards may be used for general objects [Jeschke 2005]. Billboards may also be adapted to be (more) view independent [Schaufler 1998, Décoret 2003]. In terms of compromising between geometry and texture, billboards chooses clearly to favor textures. More balanced solutions include TDM (Textured Depth Meshes) (e.g. [Décoret 1999, Jeschke 2002, Wilson 2003, Ghiletiuc 2013]). However, these representations are not progressive by default. Moreover, the choice of the representation should probably also take into account the considered setting/application, for example whether the

viewpoint changes, the resolution of/distance to the object, and also the resources of the user (rendering capacities and bandwidth). In a sense, we do ask the question "how much 3D information is needed/affordable?" in this context. A depth image may be enough, for example for (real time or interactive time) augmenting a fixed viewpoint video. A full 3D geometric model is probably needed is a professional artist wants to study a statue.

Chapter 5

Conclusion and perspectives

In this document, we have used different models, carefully chosen for a given application. In the future, it will be interesting to consider multi-model representations. A joint representation combining subdivision surfaces and parametric models is already used in the CATIA modeling software. For simplifying edition, in particular topological changes, a subdivision model is used during edition. As soon as the designer stops editing, the model is converted into an approximating NURBS based parametric model. Such a multi-model representation allows to benefit from advantages of both models.
We may consider a representation combining an implicit and a parametric representation; we could inter-operate implicit and parametric models based on a skeleton. Sampling, and texture mapping could be done on the parametric model. Editing could be done using the implicit representation for large changes, in particular those implying topological changes; for local changes, using control points (or weight) from the parametric representation will be more adapted. The implicit representation could be used for detecting intersection between objects. The skeleton by itself is relevant for animation. However, a conversion method to get both models (exactly or approximately) is necessary. Converting from parametric to implicit form is called implicitization. This problem has also been studied for long, and is still an active research field [Dokken 2006, Aigner 2012]. But, the implicit representation we consider are not necessarily algebraic. Some authors have proposed both implicit and rational parametric representations for canal surfaces, where the surface is the envelop of a family of spheres [Dohm 2009]. Sweep surfaces have also been considered as well, and can lead to implicit representation [Schmidt 2005] and parametric representations, up to an approximation error [Elber 1997]. Skinning has also been proposed as a mix of implicit and explicit representation but no notion of skeleton is used there [Vaillant 2013]. We may want to create a parameterization anchored on the skeleton. In particular, we could create a parametric tensor product surface within a given tolerance of the implicit model; we may use the skeleton to give one of the tensor product direction, thus the other direction would follow a section curve. Related work on sweep surfaces is then relevant, but only relevant for the linear part of the skeleton (no junctions).

Another perspective is to provide more general progressive models. The principle used for the progressive representation of plants (section 4.3) could be generalized for parametric or mesh models. Indeed, when compressing branching system using the proposed progressive representation we obtain very light models (20 KB) for transmitting to mobile devices. Using the similarity analysis proposed in section 3.3.4.1, we could associate parts that are similar, or close to be similar, and define a common model, instantiated several times. In case of approximate similarity, the details could be coded and quantified. This representation would be compact and progressive, and therefore adapted for streaming. Moreover, edition or comparison of 3D models could also be performed more easily on this representation than on the original representation.

The two proposed perspectives above imply only 3D models. Another perspective, mentioned in the conclusion of the last chapter, is to consider textured models, and, as such, combine

geometric information and attributes (color or textures). A continuum of representation exists, but the challenge here is to get some multi-resolution or progressive representation, for application like streaming, or adaptation to resources.

In Chapter 3, we have considered associating 3D content and some textual content to ease the navigation around 3D objects. This same idea has been applied in the context of Wikipedia, using the reconstruction from a large sequence of images, in the recent paper *Wikipedia 3D* [Russell 2013]. Using 3D together with other multimedia content is probably the way to go, as 3D content by itself is sometimes cumbersome to interact with. In this context, benefiting from the experience of other users, using crowdsourcing techniques, would certainly be an interesting way to explore.

Bibliography

[Adamson 2003] Anders Adamson and Marc Alexa. *Approximating and Intersecting Surfaces from Points*. In Eurographics/ACM SIGGRAPH Symposium on Geometry Processing, pages 245–254, 2003. (Cited on page 47.)

[Aigner 2012] Martin Aigner, Bert Jüttler and Adrien Poteaux. *Approximate implicitization of space curves*. In Numerical and Symbolic Scientific Computing, pages 1–19. Springer, 2012. (Cited on page 133.)

[Alexa 2003] Marc Alexa, Johannes Behr, Daniel Cohen-Or, Shachar Fleishman, David Levin and Claudio T. Silva. *Computing and Rendering Point Set Surfaces*. IEEE Transactions on Visualization and Computer Graphics, vol. 9, no. 1, pages 3–15, 2003. (Cited on page 44.)

[Aliaga 1999] Daniel Aliaga, Jon Cohen, Andrew Wilson, Eric Baker, Hansong Zhang, Carl Erikson, Kenny Hoff, Tom Hudson, Wolfgang Stuerzlinger, Rui Bastos, Mary Whitton, Fred Brooks and Dinesh Manocha. *MMR: an interactive massive model rendering system using geometric and image-based acceleration*. In Proceedings of I3D '99, pages 199–206, 1999. (Cited on page 11.)

[Alliez 2001] Pierre Alliez and Mathieu Desbrun. *Progressive compression for lossless transmission of triangle meshes*. In SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques, pages 195–202, New York, NY, USA, August 2001. (Cited on pages 91 and 93.)

[Alliez 2005] Pierre Alliez and Craig Gotsman. Advances in Multiresolution for Geometric Modelling., chapitre Recent Advances in Compression of 3D Meshes, pages 3–26. N.A. Dodgson and M.S. Floater and M.A. Sabin. Springer-Verlag, 2005. (Cited on page 91.)

[Amenta 1998] Nina Amenta, Marshall Bern and David Eppstein. *The crust and the β-skeleton: Combinatorial curve reconstruction*. Graphical models and image processing, vol. 60, no. 2, pages 125–135, 1998. (Cited on page 22.)

[Attali 2009] Dominique Attali, Jean-Daniel Boissonnat and Herbert Edelsbrunner. *Stability and computation of medial axes-a state-of-the-art report*. In Mathematical foundations of scientific visualization, computer graphics, and massive data exploration, pages 109–125. Springer, 2009. (Cited on page 22.)

[Bajaj 1997] Chandrajit Bajaj, HY Lee, R Merkert and Valerio Pascucci. *NURBS based B-rep models for macromolecules and their properties*. In Proceedings of the fourth ACM symposium on Solid modeling and applications, pages 217–228. ACM, 1997. (Cited on page 54.)

[Bajaj 1999] Chandrajit L Bajaj, Valerio Pascucci and Guozhong Zhuang. *Single-resolution compression of arbitrary triangular meshes with properties*. In Data Compression Conference, 1999. Proceedings. DCC'99, pages 247–256. IEEE, 1999. (Cited on page 91.)

[Bao 2006] P. Bao and D. Gourlay. *A framework for remote rendering of 3-D scenes on limited mobile devices.* IEEE Transactions on Multimedia, vol. 8, no. 2, pages 382–389, April 2006. (Cited on page 126.)

[Barsky 1984] Brian A. Barsky and Anthony D. DeRose. *Geometric continuity of parametric curves.* Rapport technique, UC Berkeley, 1984. (Cited on page 9.)

[Barthe 2011] Loic Barthe. Modèles pour la cr ÌĄeation interactive et intuitive d'objets tridimensionnels. Habilitation à diriger les recherches, 2011. (Cited on page 10.)

[Behr 2010] Johannes Behr, Yvonne Jung, Jens Keil, Timm Drevensek, Michael Zoellner, Peter Eschler and Dieter Fellner. *A scalable architecture for the HTML5/X3D integration model X3DOM.* In Web3D. ACM, 2010. (Cited on page 71.)

[Behrendt 2005] Stephan Behrendt, Carsten Colditz, Oliver Franzke, Johannes Kopf and Oliver Deussen. *Realistic real-time rendering of landscapes using billboard clouds.* Computer Graphics Forum, vol. 24, no. 3, pages 507–516, September 2005. (Cited on pages 94 and 107.)

[Berner 2008] Alexander Berner, Martin Bokeloh, Michael Wand, Andreas Schilling and H-P Seidel. *A graph-based approach to symmetry detection.* In Proceedings of the Fifth Eurographics/IEEE VGTC conference on Point-Based Graphics, pages 1–8. Eurographics Association, 2008. (Cited on page 55.)

[Besl 1986] Paul J Besl and Ramesh C Jain. *Invariant surface characteristics for 3D object recognition in range images.* Computer vision, graphics, and image processing, vol. 33, no. 1, pages 33–80, 1986. (Cited on page 85.)

[Bey 2012] Aurelien Bey, Raphaëlle Chaine, Raphael Marc and Guillaume Thibault. *Effective shapes generation for Bayesian CAD model reconstruction.* In Proceedings of the 5th Eurographics conference on 3D Object Retrieval, pages 63–66. Eurographics Association, 2012. (Cited on page 16.)

[Bloomenthal 1985] Jules Bloomenthal. *Modeling the mighty maple.* Proceedings of SIGGRAPH 1985, pages 305–311, July 1985. (Cited on page 17.)

[Bloomenthal 1997] J. Bloomenthal, editeur. Introduction to implicit surfaces. Morgan Kaufmann, 1997. (Cited on page 10.)

[Blum 1967] Harry Blum. *A transformation for extraction new descriptors of shape, models for the perception of speech and visual form.* Models for the Perception of Speech and Visual Form, pages 362–380, 1967. (Cited on page 22.)

[Bogacki 1995] Przemyslaw Bogacki, Stanley E. Weinstein and Yuesheng Xu. *Degree reduction of Bézier curves by uniform approximation with endpoint interpolation.* Computer-Aided Design, vol. 27, no. 9, pages 651–661, 1995. (Cited on page 99.)

[Bokeloh 2009] Martin Bokeloh, Alexander Berner, Michael Wand, H-P Seidel and Andreas Schilling. *Symmetry detection using feature lines.* In Computer Graphics Forum, volume 28, pages 697–706. Wiley Online Library, 2009. (Cited on page 55.)

[Bora 1990] PK Bora, YV Venkatesh and KR Ramakrishnan. *Shape from shading using discrete polynomials.* In Proceedings of the XVI Annual Convention and Exhibition of the IEEE. IEEE, 1990. (Cited on page 36.)

[Botsch 2005] Mario Botsch, Alexander Hornung, Matthias Zwicker and Leif Kobbelt. *High-Quality Surface Splatting on Today's GPUs.* In Eurographics Symposium on Point-Based Graphics 2005, pages 17–24, 2005. (Cited on page 45.)

[Boudon 2006] F. Boudon, A. Meyer and C. Godin. *Survey on Computer Representations of Trees for Realistic and Efficient Rendering.* Research report, no. RR-LIRIS-2006-003, 2006. (Cited on page 17.)

[Boudon 2010] F. Boudon, T. Cokelaer, C. Pradal and C. Godin. *L-Py, an open L-systems framework in Python.* 6th International Workshop on Functional-Structural Plant Models, pages 116–119, 2010. (Cited on page 26.)

[Boukerche 2007] A. Boukerche, R. Werner and N. Pazzi. *A Peer-to-Peer Approach for Remote Rendering and Image Streaming in Walkthrough Applications.* In Communications, 2007. ICC '07. IEEE International Conference on, pages 1692–1697, 2007. (Cited on pages 126 and 130.)

[Buehler 2001] Chris Buehler, Michael Bosse, Leonard McMillan, Steven Gortler and Michael Cohen. *Unstructured lumigraph rendering.* In Proceedings of SIGGRAPH '01, pages 425–432, 2001. (Cited on page 11.)

[Burelli 2010] Paolo Burelli and Georgios N Yannakakis. *Combining local and global optimisation for virtual camera control.* In Computational Intelligence and Games (CIG), 2010 IEEE Symposium on, pages 403–410. IEEE, 2010. (Cited on page 117.)

[C. Yuksel 2011] J. Keyser C. Yuksel S. Schaefer. *Parameterization and applications of Catmull-Rom curves.* Computer Aided Design, vol. 43, pages 747–755, 2011. (Cited on page 118.)

[Cardone 2006] Antonio Cardone, Satyandra K Gupta, Abhijit Deshmukh and Mukul Karnik. *Machining feature-based similarity assessment algorithms for prismatic machined parts.* Computer-Aided Design, vol. 38, no. 9, pages 954–972, 2006. (Cited on page 54.)

[Carlier 2010] Axel Carlier, Vincent Charvillat, Wei Tsang Ooi, Romulus Grigoras and Geraldine Morin. *Crowdsourced automatic zoom and scroll for video retargeting.* In in *ACM MM10*, pages 201–210, 2010. (Cited on page 69.)

[Catmull 1974] E. Catmull and R. Rom. *A class of local interpolating splines.* Computer aided geometric design, pages 317–326, 1974. (Cited on page 26.)

[Catmull 1978] Edwin Catmull and James Clark. *Recursively generated B-spline surfaces on arbitrary topological meshes.* Computer-aided design, vol. 10, no. 6, pages 350–355, 1978. (Cited on page 10.)

[Caumon 2003] Guillaume Caumon, Charles H Sword Jr and Jean-Laurent Mallet. *Constrained modifications of non-manifold b-reps.* In Proceedings of the eighth ACM symposium on Solid modeling and applications, pages 310–315. ACM, 2003. (Cited on page 54.)

[Chang 2002] C.-F. Chang and S.-H. Ger. *Enhancing 3D graphics on mobile devices by image-based rendering.* In IEEE PCM '02, 2002. (Cited on page 126.)

[Changa 2011] Yao-Jen Changa, Shu-Fang Chenb and Jun-Da Huangc. *A Kinect-based system for physical rehabilitation: A pilot study for young adults with motor disabilities.* Research in Developmental Disabilities, vol. 32, no. 6, pages 2566–2570, 2011. (Cited on page 7.)

[Chaubert-Pereira 2010] Florence Chaubert-Pereira, Yann Guédon, Christian Lavergne and Catherine Trottier. *Markov and semi-Markov switching linear mixed models used to identify forest tree growth components.* Biometrics, vol. 66, page 753, 2010. (Cited on page 17.)

[Chen 2001] Baoquan Chen and Minh Xuan Nguyen. *POP: a hybrid point and polygon rendering system for large data.* In VIS '01: Proceedings of the Conference on Visualization '01, pages 45–52. IEEE Computer Society, 2001. (Cited on page 91.)

[Chen 2003] Zhihua Chen, Bobby Bodenheimer and J. Fritz Barnes. *Robust transmission of 3D geometry over lossy networks.* In Web3D '03: Proceeding of the eighth international conference on 3D Web technology, pages 161–ff, New York, NY, USA, 2003. ACM Press. (Cited on page 93.)

[Chen 2013] Tao Chen, Zhe Zhu, Ariel Shamir, Herzliya, Shi-Min Hu and Daniel Cohen-Or. *3-Sweep: Extracting Editable Objects from a Single Photo.* In SIGGRAPH Asia '13: ACM SIGGRAPH Asia 2013 papers, 2013. (Cited on page 39.)

[Cheng 2004] L. Cheng, A. Bhushan, R. Pajarola and M. E. Zarki. *Real-time 3D graphics streaming using MPEG-4.* In Proc. of the IEEE/ACM Workshop on Broadband Wireless Services and Applications, 2004. (Cited on page 122.)

[Cheng 2007] Wei Cheng, Wei Tsang Ooi, Sebastien Mondet, Romulus Grigoras and Géraldine Morin. *An analytical model for progressive mesh streaming.* In Proceedings of the 15th international conference on Multimedia, MULTIMEDIA '07, pages 737–746. ACM, 2007. (Cited on pages 11, 90 and 108.)

[Cheng 2011] Wei Cheng, Wei Tsang Ooi, Sebastien Mondet, Romulus Grigoras and Géraldine Morin. *Modeling progressive mesh streaming: Does data dependency matter?* ACM Trans. Multimedia Comput. Commun. Appl., vol. 7, no. 2, pages 10:1–10:24, 2011. (Cited on pages 11, 108 and 113.)

[Chittaro 2002] Luca Chittaro and Roberto Ranon. *Dynamic generation of personalized VRML content: a general approach and its application to e-commerce.* In in proceedings of Web3D '02, pages 145–154, 2002. (Cited on page 69.)

[Chittaro 2007] Luca Chittaro and Roberto Ranon. The adaptive web, chapitre Adaptive 3D web sites, pages 433–462. Springer-Verlag, 2007. (Cited on page 70.)

[Choppin 2012] Simon Choppin and Jonathan Wheat. *Marker-less tracking of human movement using Microsoft Kinect.* In ISBS-Conference Proceedings Archive, volume 1, 2012. (Cited on pages 7 and 8.)

[Chu 2006] Chih-Hsing Chu and Yung-Chang Hsu. *Similarity assessment of 3D mechanical components for design reuse.* Robotics and Computer-Integrated Manufacturing, vol. 22, no. 4, pages 332–341, 2006. (Cited on page 54.)

[Cohen-Or 1999] Daniel Cohen-Or, David Levin and Offir Remez. *Progressive compression of arbitrary triangular meshes.* In VIS '99: Proceedings of the conference on Visualization '99, pages 67–72, Los Alamitos, CA, USA, October 1999. (Cited on page 93.)

[Comaniciu 2002] Dorin Comaniciu and Peter Meer. *Mean Shift: A Robust Approach Toward Feature Space Analysis.* IEEE Trans. Pattern Anal. Mach. Intell., vol. 24, May 2002. (Cited on page 71.)

[Cornea 2005] Nicu D. Cornea, Deborah Silver, Xiaosong Yuan and Raman Balasubramanian. *Computing hierarchical curve-skeletons of 3d objects.* The Visual Computer, vol. 21, 2005. (Cited on page 23.)

[Cornea 2007] Nicu D. Cornea, Deborah Silver and Patrick Min. *Curve-Skeleton Properties, Applications, and Algorithms.* IEEE Transactions on Visualization and Computer Graphics, vol. 13, no. 3, pages 530–548, May 2007. (Cited on page 23.)

[Costes 2003] Evelyne Costes, Hervé Sinoquet, Jean-Jacques Kelner and Christophe Godin. *Exploring within-tree architectural development of two apple treecultivars over 6 years.* Annals of Botany, vol. 91, pages 91–104, 2003. (Cited on page 104.)

[Courteille 2004] Frederic Courteille, Alain Crouzil, J-D Durou and Pierre Gurdjos. *Towards shape from shading under realistic photographic conditions.* In Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on, volume 2, pages 277–280. IEEE, 2004. (Cited on page 36.)

[Courteille 2006a] Frédéric Courteille, Jean-Denis Durou and GÎĄeraldine Morin. *A Global Solution to the SFS Problem Using B-spline Surface and Simulated Annealing.* In International Conference on Pattern Recognition (ICPR), volume 2. IEEE Computer Society, 2006. (Cited on pages 11, 16 and 36.)

[Courteille 2006b] Frédéric Courteille, Jean-Denis Durou and GÎĄeraldine Morin. *Shape from Shading : Reconstruction using a B-spline Model.* International Conference on Curves and Surfaces, 2006. (Cited on page 11.)

[Cruz-Neira 1992] Carolina Cruz-Neira, Daniel J. Sandin, Thomas A. DeFanti, Robert V. Kenyon and John C. Hart. *The CAVE: audio visual experience automatic virtual environment.* Commun. ACM, vol. 35, no. 6, pages 64–72, 1992. (Cited on page 4.)

[Cruz-Neira 1993] Carolina Cruz-Neira, Daniel J. Sandin and Thomas A. DeFanti. *Surround-screen projection-based virtual reality: the design and implementation of the CAVE.* In Proceedings of the 20th annual conference on Computer graphics and interactive techniques, SIGGRAPH '93, pages 135–142, 1993. (Cited on page 4.)

[Dang 2012] Quoc-Viet Dang, Sandrine Mouysset and GÎĄeraldine Morin. *Détection de Similarités de Surfaces Paramétriques.* Revue Electronique Francophone d'Informatique Graphique, vol. 6, pages 50–58, 2012. (Cited on pages 11 and 42.)

[Dang 2013] Quoc-Viet Dang, Sandrine Mouysset and GÎĄeraldine Morin. *Similarity Detection for Free-Form Parametric Models.* WSCG, 2013. (Cited on pages 11 and 42.)

[de Reffye 1988] Phillippe de Reffye, Claude Edelin, Jean Françon, Marc Jaeger and Claude Puech. *Plant models faithful to botanical structure and development.* SIGGRAPH Comput. Graph., vol. 22, no. 4, pages 151–158, June 1988. (Cited on page 17.)

[Debevec 1998] Paul Debevec, Yizhou Yu and George Boshokov. *Efficient View-Dependent Image-Based Rendering with Projective Texture-Mapping.* Rapport technique, Berkeley, CA, USA, 1998. (Cited on page 11.)

[Decaudin 2004] Philippe Decaudin and Fabrice Neyret. *Rendering Forest Scenes in Real-Time.* In Proceedings of the 15th Eurographics Symposium on Rendering, pages 93–102, Norköping, Sweden, June 2004. (Cited on pages 94 and 107.)

[Décoret 1999] Xavier Décoret, Gernot Schaufler, François Sillion and Julie Dorsey. *Multi-Layered Impostors for Accelerated Rendering.* Computer Graphics Forum, vol. 18, no. 3, pages 61–73, September 1999. (Cited on page 131.)

[Décoret 2003] Xavier Décoret, Frédo Durand, François X. Sillion and Julie Dorsey. *Billboard clouds for extreme model simplification.* ACM Trans. Graph., vol. 22, no. 3, pages 689–696, July 2003. (Cited on page 131.)

[Deering 1995] Michael Deering. *Geometry compression.* In Proceedings of the 22nd annual conference on Computer graphics and interactive techniques, pages 13–20. ACM, 1995. (Cited on page 91.)

[Dehais 2006] Christophe Dehais, Géraldine Morin and Vincent Charvillat. *3D Visual Tracking Using a Point- based Model.* In Vision, Modeling and Visualization (VMV), 2006. (Cited on pages 11 and 42.)

[Dehais 2008] Christophe Dehais. *Contributions pour les applications de réalité augmentée : Suivi visuel et recalage 2D; Suivi d'objets 3D représentés par des modèles par points.* PhD thesis, Institut National Polytechnique de Toulouse, 2008. (Cited on page 42.)

[Dehais 2010] Christophe Dehais, Géraldine Morin and Vincent Charvillat. *From Rendering to Tracking Point- based 3D Models.* Image and Vision Computing, 2010. (Cited on pages 11 and 42.)

[Delaunay 2007a] Xavier Delaunay, Marie Chabert, Géraldine Morin and Vincent Charvillat. *Bit-plane analysis and contexts combining of JPEG2000 contexts for on-board satellite image compression.* In Acoustics, Speech and Signal Processing, 2007. ICASSP 2007. IEEE International Conference on, volume 1, pages I–1057. IEEE, 2007. (Cited on page 13.)

[Delaunay 2007b] Xavier Delaunay, Marie Chabert, Géraldine Morin and Carole Thiebaut. *Post-transformée dans le domaine ondelettes appliquée à la compression d'images satellite.* In 21 ° Colloque GRETSI, Troyes, FRA, 11-14 septembre 2007. GRETSI, Groupe d'Etudes du Traitement du Signal et des Images, 2007. (Cited on page 13.)

[Delaunay 2008a] Xavier Delaunay, Marie Chabert, Vincent Charvillat, Géraldine Morin and Rosario Ruiloba. *Satellite image compression by directional decorrelation of wavelet coefficients.* In Acoustics, Speech and Signal Processing, 2008. ICASSP 2008. IEEE International Conference on, pages 1193–1196. IEEE, 2008. (Cited on page 13.)

[Delaunay 2008b] Xavier Delaunay, Carole Thiebaut, Emmanuel Christophe, Rosa Ruiloba, Marie Chabert, Vincent Charvillat and Géraldine Morin. *Lossy compression by post-transforms in the wavelet domain.* In On-Board Payload Data Compression Workshop, 2008. (Cited on page 13.)

[Delaunay 2010] Xavier Delaunay, Marie Chabert, Vincent Charvillat and Géraldine Morin. *Satellite image compression by post-transforms in the wavelet domain.* Signal Processing, vol. 90, no. 2, pages 599–610, 2010. (Cited on page 13.)

[deRose 2000] Tony deRose. *Subdivision Surfaces in Feature Films.* In Mathematical methods for curves and surfaces, 2000. (Cited on pages 3 and 4.)

[Deussen 2005] Oliver Deussen and Bernd Lintermann. Digital Design of Nature: Computer Generated Plants and Organics. Springer-Verlag, 2005. (Cited on pages 17, 94 and 107.)

[Devillers 2000] Olivier Devillers and Pierre-Marie Gandoin. *Geometric Compression for Interactive Transmission.* Rapport technique, INRIA - Sophia Antipolis, 2000. (Cited on page 91.)

[Dimas 1999] E Dimas and D Briassoulis. *3D geometric modelling based on NURBS: a review.* Advances in Engineering Software, vol. 30, no. 9, pages 741–751, 1999. (Cited on page 54.)

[Dohm 2009] Marc Dohm and Severinas Zube. *The implicit equation of a canal surface.* Journal of Symbolic Computation, vol. 44, no. 2, pages 111–130, 2009. (Cited on page 133.)

[Dokken 2006] T Dokken and Jan B Thomassen. *Weak approximate implicitization.* In Shape Modeling and Applications, 2006. SMI 2006. IEEE International Conference on, pages 31–31. IEEE, 2006. (Cited on page 133.)

[Doran 2009] Andra Doran, Sebastien Mondet, Romulus Grigoras, Géraldine Morin, Wei Tsang Ooi and Frédéric Boudon. *A demonstration of MobiTree: progressive 3D tree models streaming on mobile clients.* In ACM Multimedia, 2009. (Cited on pages 12, 90 and 121.)

[Drummond 2002] Tom Drummond and Roberto Cipolla. *Real-Time Visual Tracking of Complex Structures.* IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 24, no. 7, pages 932–946, July 2002. (Cited on pages 43, 48 and 50.)

[Dutagaci 2010] Helin Dutagaci, Chun Pan Cheung and Afzal Godil. *A benchmark for best view selection of 3D objects.* In Proceedings of the ACM Workshop on 3D Object Retrieval, 3DOR '10, 2010. (Cited on page 117.)

[Dyn 2002] Nira Dyn and David Levin. *Subdivision schemes in geometric modelling.* Acta Numerica, vol. 11, no. 0, pages 73–144, 2002. (Cited on page 10.)

[Elber 1997] Gershon Elber. *Global error bounds and amelioration of sweep surfaces.* Computer-Aided Design, vol. 29, no. 6, pages 441–447, 1997. (Cited on page 133.)

[Farin 2002a] Gerald Farin. Curves and surfaces for CAGD: a practical guide. Morgan Kaufmann Publishers Inc., 2002. (Cited on pages 9, 56 and 101.)

[Farin 2002b] Gerald Farin. *A history of curves and surfaces in CAGD.* Handbook of Computer Aided Geometric Design, pages 1–23, 2002. (Cited on page 3.)

[Fleishman 2003] Shachar Fleishman, Marc Alexa, Daniel Cohen-Or and Claudio T. Silva. *Progressive Point set surfaces.* ACM Transactions on Computer Graphics, vol. 22, no. 4, 2003. (Cited on page 91.)

[Foley 1987] T. Foley. *Interpolation with interval and point tension controls using cubic weighted Îį-spline.* ACM Trans. on Math. Software,, vol. 13, pages 68–96, 1987. (Cited on page 9.)

[Freitag 2002] Lori A Freitag and Patrick M Knupp. *Tetrahedral mesh improvement via optimization of the element condition number.* International Journal for Numerical Methods in Engineering, vol. 53, no. 6, pages 1377–1391, 2002. (Cited on page 9.)

[Ghiletiuc 2013] Johannes Ghiletiuc, Markus Färber and Beat Brüderlin. *Real-time remote rendering of large 3D models on smartphones using multi-layered impostors.* In Proceedings of the 6th International Conference on Computer Vision / Computer Graphics Collaboration Techniques and Applications, MIRAGE '13, pages 14:1–14:8, 2013. (Cited on page 131.)

[Goesele 2007] Michael Goesele, Noah Snavely, Brian Curless, Hugues Hoppe and Steven M Seitz. *Multi-view stereo for community photo collections.* In Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on, pages 1–8. IEEE, 2007. (Cited on page 16.)

[Goldman 2002] Ron Goldman. Pyramid algorithms: A dynamic programming approach to curves and surfaces for geometric modeling. Morgan Kaufmann, 2002. (Cited on page 9.)

[Gortler 1996] Steven J. Gortler, Radek Grzeszczuk, Richard Szeliski and Michael F. Cohen. *The lumigraph.* In Proceedings of SIGGRAPH '96, pages 43–54, 1996. (Cited on page 11.)

[Gross 2007] Markus Gross and Hanspeter Pfister. Point-based graphics. Morgan Kauffman, 2007. (Cited on page 43.)

[Gross 2011] Markus Gross and Hanspeter Pfister. Point-based graphics. Morgan Kaufmann, 2011. (Cited on page 10.)

[Guénard 2009] Jérôme Guénard, Géraldine Morin, Pierre Gurdjos and Vincen Charvillat. *De la Reconstruction de Quadriques de Révolution à partir d'Images à la Complémentation d'Objets Naturels.* In 22èmes journées de l'Association Francophone d'Informatique Graphique, 2009. (Cited on page 11.)

[Guénard 2010] Jérôme Guénard, Charlotte Giron, Géraldine Morin, Pierre Gurdjos, Frédéric Boudon, Vincent Charvillat*et al. Modélisation de vignes à partir d'une séquence d'images.* In 23èmes journées de l'Association Francophone d'Informatique Graphique, 2010. (Cited on pages 11 and 16.)

[Guénard 2011] Jérôme Guénard, Géraldine Morin, Frédéric Boudon, Pierre Gurdjos and Vincent Charvillat. *Reconstruction de modèles virtuels de vignes à partir d'images.* In ORASIS-Congrès des jeunes chercheurs en vision par ordinateur, 2011. (Cited on pages 11 and 16.)

[Guénard 2012] Jérôme Guénard, Géraldine Morin and Vincent Charvillat. *Realistic Plant Modeling from Images based on Analysis-by-Synthesis.* In International Conference on Mathematical Methods for Curves and Surfaces, 2012. (Cited on pages 11 and 16.)

[Guénard 2013a] Jérôme Guénard. *Synthèse de modèles de plantes et reconstructions de baies à partir d'images*. PhD thesis, University of Toulouse, 2013. (Cited on pages 16, 21, 24 and 34.)

[Guénard 2013b] Jérôme Guénard, GĨĄeraldine Morin, Frédéric Boudon and Vincent Charvillat. *Reconstructing Plants in 3D from a Single Image using Analysis-by-Synthesis*. In International Symposium on Visual Computing, 2013. (Cited on pages 11 and 16.)

[Guennebaud 2003] Gael Guennebaud, Mathias Paulin*et al.* *Efficient Screen Space Approach for Hardware Accelerated Surfel Rendering.* In VMV, volume 20003, pages 1–10, 2003. (Cited on page 10.)

[Guennebaud 2004] Gael Guennebaud, Loïc Barthe and Mathias Paulin. *Deferred splatting*. In Computer Graphics Forum, volume 23, pages 653–660. Wiley Online Library, 2004. (Cited on pages 10 and 45.)

[Gumhold 1999] Stefan Gumhold, Stefan Guthe and Wolfgang Straßer. *Tetrahedral mesh compression with the cut-border machine*. In Proceedings of the conference on Visualization'99: celebrating ten years, pages 51–58. IEEE Computer Society Press, 1999. (Cited on page 9.)

[Harris 1988] C. Harris and M. Stephens. *A Combined Corner and Edge Detection*. In Proceedings of The Fourth Alvey Vision Conference, pages 147–151, 1988. (Cited on page 46.)

[Hartley 2003] Richard Hartley and Andrew Zisserman. Multiple view geometry in computer vision. Cambridge University Press, New York, NY, USA, 2003. (Cited on page 46.)

[Hilaga 2001] Masaki Hilaga, Yoshihisa Shinagawa, Taku Kohmura and Tosiyasu L Kunii. *Topology matching for fully automatic similarity estimation of 3D shapes*. In Proceedings of the 28th annual conference on Computer graphics and interactive techniques, pages 203–212. ACM, 2001. (Cited on page 54.)

[Hoppe 1996] Hugues Hoppe. *Progressive Meshes*. Computer Graphics, vol. 30, no. Annual Conference Series, pages 99–108, August 1996. (Cited on pages 11 and 92.)

[Horn 1989] Berthold K. P. Horn. *Obtaining shape from shading information*. In Berthold K. P. Horn and Michael J. Brooks, editeurs, Shape from shading, pages 123–171. MIT Press, Cambridge, MA, USA, 1989. (Cited on page 36.)

[Hughes 2002] Stephen Hughes, Peter Brusilovsky and Michael Lewis. *Adaptive navigation support in 3D e-commerce activities*. In Workshop on Recommendation and Personalization in eCommerce, 2002. (Cited on page 69.)

[Isenburg 2003] Martin Isenburg and Stefan Gumhold. *Out-of-core compression for gigantic polygon meshes*. In ACM Transactions on Graphics (TOG), volume 22, pages 935–942. ACM, 2003. (Cited on page 91.)

[Iyer 2005] Natraj Iyer, Subramaniam Jayanti, Kuiyang Lou, Yagnanarayanan Kalyanaraman and Karthik Ramani. *Three-dimensional shape searching: state-of-the-art review and future trends*. Computer-Aided Design, vol. 37, no. 5, pages 509–530, 2005. (Cited on page 54.)

[Jeschke 2002] Stefan Jeschke and Michael Wimmer. *Textured depth meshes for real-time rendering of arbitrary scenes.* In Proceedings of EGWR '02, pages 181–190, 2002. (Cited on page 131.)

[Jeschke 2005] Stefan Jeschke, Michael Wimmer and Werner Purgathofer. *Image-based Representations for Accelerated Rendering of Complex Scenes.* In Y. Chrysanthou and M. Magnor, editeurs, EUROGRAPHICS 2005 State of the Art Reports, pages 1–20. EUROGRAPHICS, aug 2005. (Cited on pages 11 and 131.)

[Jie Feng 2004] H. Z. Jie Feng. *Efficient view-dependent LOD control for large 3D unclosed mesh models of environments.* In IEEE ICRA, pages 2723–2728, 2004. (Cited on page 130.)

[Johnson 1965] E.A. Johnson. *Touch Display - A novel input/output device for computers.* Electronics letters, vol. 1, no. 8, pages 271–277, 1965. (Cited on page 7.)

[Kamada 1988] Tomihisa Kamada and Satoru Kawai. *A simple method for computing general position in displaying three-dimensional objects.* Computer Vision, Graphics, and Image Processing, vol. 41, no. 1, pages 43–56, 1988. (Cited on page 117.)

[Kazhdan 2004] Michael Kazhdan, Bernard Chazelle, David Dobkin, Thomas Funkhouser and Szymon Rusinkiewicz. *A reflective symmetry descriptor for 3D models.* Algorithmica, vol. 38, no. 1, pages 201–225, 2004. (Cited on page 55.)

[Kim 1997] Bang-Hwan Kim and Rae-Hong Park. *Shape from shading and photometric stereo using surface approximation by legendre polynomials.* Computer Vision and Image Understanding, vol. 66, no. 3, pages 255–270, 1997. (Cited on page 36.)

[Kim 2004] HyungSeok Kim, Chris Joslin, Thomas Di Giacomo, Stephane Garchery and Nadia Magnenat-Thalmann. *Adaptation mechanism for three dimensional content within the mpeg-21 framework.* In Computer Graphics International, 2004. (Cited on page 69.)

[Kobbelt 2004] Leif Kobbelt and Mario Botsch. *A Survey of Point-Based Techniques in Computer Graphics.* Computers and Graphics, vol. 28, no. 6, pages 801–814, 2004. (Cited on page 91.)

[Kollnig 1997] Henner Kollnig and Hans-Hellmut Nagel. *3D Pose Estimation by Directly Matching Polyhedral Models to Gray Value Gradients.* Int. J. Comput. Vision, vol. 23, no. 3, pages 283–302, 1997. (Cited on page 43.)

[Laga 2010] Hamid Laga. *Semantics-driven approach for automatic selection of best views of 3d shapes.* In Eurographics conference on 3D Object Retrieval, 2010. (Cited on page 70.)

[Latecki 1999] Longin Jan Latecki and Rolf Lakamper. *Polygon Evolution by Vertex Deletion.* In Scale-Space Theories in Computer Vision. Proc. of Int. Conf. on Scale-Space'99, volume LNCS 1682, Corfu, pages 398–409. Springer, 1999. (Cited on page 24.)

[Lee 1996] Kyoung Mu Lee and C-C Jay Kuo. *Shape from photometric ratio and stereo.* Journal of Visual Communication and Image Representation, vol. 7, no. 2, pages 155–162, 1996. (Cited on page 36.)

[Lee 2009] Ho Lee, Guillaume Lavoué and Florent Dupont. *Adaptive Coarse-to-Fine Quantization for Optimizing Rate-distortion of Progressive Mesh Compression.* In VMV, pages 73–82, 2009. (Cited on page 91.)

[Lei 2004] Y. Lei, D. Jiang Z. andChen and H. Bao. *Image-based walkthrough over Internet on mobile devices.* In Grid and Cooperative Computing, pages 728–735, 2004. (Cited on pages 126 and 130.)

[Levin 2003] David Levin. *Mesh-independent surface interpolation.* Geometric Modeling for Scientific Visualization, pages 37–49, 2003. (Cited on page 44.)

[Levoy 1985] M. Levoy and T. Whitted. *The use of points as display primitives.* Rapport technique, Computer science department, North Carolina University, 1985. (Cited on pages 6 and 10.)

[Levoy 1996] Marc Levoy and Pat Hanrahan. *Light field rendering.* In Proceedings of SIGGRAPH '96, pages 31–42, 1996. (Cited on page 11.)

[Levoy 2000] Marc Levoy, Kari Pulli, Brian Curless, Szymon Rusinkiewicz, David Koller, Lucas Pereira, Matt Ginzton, Sean Anderson, James Davis, Jeremy Ginsberg, Jonathan Shade and Duane Fulk. *The Digital Michelangelo Project: 3D Scanning of Large Statues.* In SIGGRAPH 2000, Computer Graphics Proceedings, pages 131–144, July 2000. (Cited on pages 15 and 88.)

[Li 2011] Chuan Li, Oliver Deussen, Yi-Zhe Song, Phil Willis and Peter Hall. *Modeling and generating moving trees from video.* In Proceedings of the 2011 SIGGRAPH Asia Conference, SA '11, pages 127:1–127:12, New York, NY, USA, 2011. ACM. (Cited on page 39.)

[Lindenmayer 1968] Aristid Lindenmayer. *Mathematical models for cellular interaction in development: Parts I and II.* Journal of Theoretical Biology, vol. 18, 1968. (Cited on page 17.)

[Lipman 2009] Y. Lipman and T Funkhouser. *Möbius voting for surface correspondence.* ACM Transactions on Graphics (TOG), page p.72, 2009. (Cited on pages 55 and 56.)

[Liu 2010] Jia Liu, Xiaopeng Zhang, Hongjun Li and Mingrui Dai. *Creation of tree models from freehand sketches by building 3D skeleton point cloud.* In Proceedings of the Entertainment for education, and 5th international conference on E-learning and games, Edutainment'10, pages 621–632, Berlin, Heidelberg, 2010. Springer-Verlag. (Cited on page 20.)

[Lluch 2005] J. Lluch, E. Gaitan R. andCamahort and R. Vivo. *Interactive three-dimensional rendering on mobile computer devices.* In ACM SIGCHI, pages 254–257, 2005. (Cited on page 130.)

[Loop 1987] Charles Loop. Smooth subdivision surfaces based on triangles. Master's thesis, University of Utah, research.microsoft.com/ cloop/thesis.pdf, 1987. (Cited on page 10.)

[Loop 2008] Charles Loop and Scott Schaefer. *Approximating Catmull-Clark subdivision surfaces with bicubic patches.* ACM Transactions on Graphics (TOG), vol. 27, no. 1, page 8, 2008. (Cited on page 6.)

[Loop 2009] Charles Loop, Scott Schaefer, Tianyun Ni and Ignacio Castaño. *Approximating subdivision surfaces with Gregory patches for hardware tessellation.* In ACM Transactions on Graphics (TOG), volume 28, page 151. ACM, 2009. (Cited on page 6.)

[Lowe 1992] David G. Lowe. *Robust model-based motion tracking through the integration of search and estimation.* Int. J. Comput. Vision, vol. 8, no. 2, pages 113–122, 1992. (Cited on page 43.)

[Ma 2010] Lujie Ma, Zhengdong Huang and Yanwei Wang. *Automatic discovery of common design structures in CAD models.* Computers & Graphics, vol. 34, no. 5, pages 545 – 555, 2010. <ce:title>CAD/GRAPHICS 2009</ce:title> <ce:title>Extended papers from the 2009 Sketch-Based Interfaces and Modeling Conference</ce:title> <ce:title>Vision, Modeling & Visualization</ce:title>. (Cited on page 54.)

[McMillan 1995] L. McMillan and Gary Bishop. *Plenoptic modeling: An image-based rendering system.* In Proceedings of SIGGRAPH'95, volume 95, pages 39–46. ACM, 1995. (Cited on pages 11, 126 and 127.)

[McMillan 1997] Leonard McMillan Jr. *An image-based approach to three-dimensional computer graphics.* PhD thesis, University of North Carolina at Chapel Hill, Chapel Hill, NC, USA, 1997. UMI Order No. GAX97-30561. (Cited on pages 126 and 130.)

[Meyer 2001] Alexandre Meyer, Fabrice Neyret and Pierre Poulin. *Interactive Rendering of Trees with Shading and Shadows.* In Proceedings of the Eurographics Workshop on Rendering Techniques, pages 183–196, June 2001. (Cited on pages 94 and 107.)

[Mitra 2013] Niloy J Mitra, Mark Pauly, Michael Wand and Duygu Ceylan. *Symmetry in 3d geometry: Extraction and applications.* In Computer Graphics Forum. Wiley Online Library, 2013. (Cited on pages 55, 56, 59, 62, 64 and 66.)

[Miyakazi 2006] D. Miyakazi, M. Kamakura, T. Higo, Y. Okamoto, R. Kawakami, T. Shiratori, A. Ikari, S. Ono, Y. Sato and M. Oya. *3D digital archive of the burghers of calais.* Lecture Notes in Computer Science, vol. 4270, page 399, 2006. (Cited on page 88.)

[Mochocki 2006] B. Mochocki, K. Lahiri and S. Cadambi. *Power analysis of mobile 3D graphics.* In Proc. of the Conf. on Design, Automation and Test in Europe, pages 502–507, 2006. (Cited on page 122.)

[Mondet 2007] Sebastien Mondet, Géraldine Morin and Romulus Grigoras. *Mise en ligne de modèles 3D echelonables basés points.* Association Française d'Informatique Graphique (AFIG), Strasbourg, France, vol. 28, no. 11, pages 05–30, 2007. (Cited on pages 10, 11, 12 and 90.)

[Mondet 2008] Sebastien Mondet, Wei Cheng, Géraldine Morin, Romulus Grigoras and Wei Tsang Ooi. *Streaming of Plants in Distributed Virtual Environments.* In Proceeding of ACM Multimedia, pages 1–10, Vancouver, Canada, 2008. (Cited on pages 12, 90, 99 and 104.)

[Mondet 2009a] Sebastien Mondet. *Adaptive Modeling and Distribution of Large Natural Scenes.* PhD thesis, Institut National Polytechnique de Toulouse, University of Toulouse, 2009. (Cited on pages 90 and 104.)

[Mondet 2009b] Sebastien Mondet, Wei Cheng, Geraldine Morin, Romulus Grigoras, Frederic Boudon and Wei Tsang Ooi. *Compact and progressive plant models for streaming in*

networked virtual environments. ACM Transactions on Multimedia Computing, Communinication, and Applications Multimedia Comput. Commun. Appl., vol. 5, no. 3, pages 21:1–21:22, 2009. (Cited on pages 12, 90, 98, 104, 108 and 113.)

[Morvan 2008] Jean-Marie Morvan. Generalized curvatures, volume 2. Springer, 2008. (Cited on page 10.)

[Mouysset 2011] S. Mouysset, J. Noailles, D. Ruiz and R. Guivarch. *On a strategy for spectral clustering with parallel computation.* In Proceedings of VECPAR, pages pp 408–420, 2011. (Cited on pages 62 and 64.)

[Munoz 2005] Enrique Munoz, Josï£¡ M. Buenaposada and Luis Baumela. *Efficient model-based 3D tracking of deformable objects.* In Proceedings of ICCV 2005, pages 877–882, Beijing, China, October 2005. (Cited on page 43.)

[Nair 2008] Vinod Nair, Josh Susskind and Geoffrey E. Hinton. *Analysis-by-Synthesis by Learning to Invert Generative Black Boxes.* ICANN, pages 971–981, 2008. (Cited on page 34.)

[Neubert 2007] Boris Neubert, Thomas Franken and Oliver Deussen. *Approximate image-based tree-modeling using particle flows.* ACM Transactions on Graphics, vol. 26, no. 3, page 88, July 2007. (Cited on pages 17, 19 and 93.)

[Nghiem 2012] Thi Phuong Nghiem, Axel Carlier, Géraldine Morin and Vincent Charvillat. *Enhancing Online 3D Products through Crowdsourcing.* In International ACM Workshop on Crowdsourcing for Multimedia, 2012. (Cited on pages 11 and 42.)

[Nghiem 2013] Thi Phuong Nghiem, Axel Carlier, Géraldine Morin and Vincent Charvillat. *Towards 3D Crowdsourcing: Easing Web3D Navigation Using User Traces.* In Eurosis 2013, 2013. (Cited on pages 11 and 42.)

[Noimark 2003] Y. Noimark and D. Cohen-Or. *Streaming scenes to MPEG-4 video-enabled devices.* In IEEE Computer Graphics and Applications, pages 58–64, 2003. (Cited on page 122.)

[Oikonomidis 2011] Iason Oikonomidis, Nikolaos Kyriazis and Antonis A Argyros. *Efficient model-based 3D tracking of hand articulations using Kinect.* In BMVC, pages 1–11, 2011. (Cited on page 7.)

[Okabe 2005] Makoto Okabe, Shigeru Owada and Takeo Igarashi. *Interactive design of botanical trees using freehand sketches and example-based editing.* Computer Graphics Forum, 2005. (Cited on pages 19 and 27.)

[Olsen 2009] Luke Olsen, Faramarz F Samavati, Mario Costa Sousa and Joaquim A Jorge. *Sketch-based modeling: A survey.* Computers & Graphics, vol. 33, no. 1, pages 85–103, 2009. (Cited on page 39.)

[Pain 2001] CC Pain, AP Umpleby, CRE De Oliveira and AJH Goddard. *Tetrahedral mesh optimisation and adaptivity for steady-state and transient finite element calculations.* Computer Methods in Applied Mechanics and Engineering, vol. 190, no. 29, pages 3771–3796, 2001. (Cited on page 9.)

[Pajarola 2000] Renato Pajarola and Jarek Rossignac. *Compressed Progressive Meshes.* IEEE Transactions on Visualization and Computer Graphics, vol. 6, no. 1, pages 79–93, January 2000. (Cited on page 93.)

[Palubicki 2009] Wojciech Palubicki, Kipp Horel, Steven Longay, Adam Runions, Brendan Lane, Radomír Měch and Przemyslaw Prusinkiewicz. *Self-organizing tree models for image synthesis.* SIGGRAPH, pages 1–10, 2009. (Cited on page 17.)

[Parisot 2004] Pascaline Parisot, Vincent Charvillat and Géraldine Morin. *Compensation de Mouvement par Maillage: Apprentissage et Maintien de la Connectivité.* In CORESA, volume 4, pages 04–26, 2004. (Cited on page 13.)

[Parisot 2005a] Pascaline Parisot, Vincent Charvillat and Géraldine Morin. *Non-rigid tracking using 2-d meshes.* In Advanced Concepts for Intelligent Vision Systems, pages 579–586. Springer, 2005. (Cited on page 13.)

[Parisot 2005b] Pascaline Parisot, Vincent Charvillat and Géraldine Morin. *Suivi par triangulation d'objets d ÌĄeformables : m ÌĄethode par apprentissage.* In ORASIS-Congrès des jeunes chercheurs en vision par ordinateur, 2005. (Cited on page 13.)

[Pauly 2002] Mark Pauly, Markus Gross and Leif P Kobbelt. *Efficient simplification of point-sampled surfaces.* In Proceedings of the conference on Visualization'02, pages 163–170. IEEE Computer Society, 2002. (Cited on page 10.)

[Pauly 2003a] Mark Pauly. *Point Primitives for Interactive Modeling and Processing of 3D Geometry.* PhD thesis, Federal Institute of Technology (ETH) of Zurich, 2003. (Cited on pages 10 and 91.)

[Pauly 2003b] Mark Pauly. *Point Primitives for Interactive Modeling and Processing of 3D Geometry.* PhD thesis, Federal Institute of Technology (ETH) of Zurich, 2003. (Cited on page 44.)

[Peleg 1997] S. Peleg and J. Herman. *Panoramic mosaics by manifold projection.* In Proceedings of CVPR '97, pages 338–, 1997. (Cited on page 11.)

[Peng 2005] Jingliang Peng, Chang-Su Kim and C-C Jay Kuo. *Technologies for 3D mesh compression: A survey.* Journal of Visual Communication and Image Representation, vol. 16, no. 6, pages 688–733, 2005. (Cited on page 92.)

[Peters 2001] Jörg Peters. *Geometric continuity.* Rapport technique, University of Florida, www.cise.ufl.edu/research/SurfLab/papers/01handbook.pdf, 2001. (Cited on page 9.)

[Peters 2008] Jörg Peters and Ulrich Reif. Subdivision surfaces. Springer, 2008. (Cited on page 10.)

[Picardi 2011] Andrea Picardi, Paolo Burelli and Georgios N. Yannakakis. *Modelling Virtual Camera Behaviour Through Player Gaze.* In International Conference On The Foundations Of Digital Games, Bordeux, France, June 2011. (Cited on page 117.)

[Piegl 1997] Les A Piegl and Wayne Tiller. The nurbs book. Springer, 1997. (Cited on page 9.)

[Plemenos 1996] Dimitri Plemenos and Madjid Benayada. *Intelligent display in scene modeling. new techniques to automatically compute good views.* In International Conference GraphiCon, volume 96, pages 1–5, 1996. (Cited on page 117.)

[Podolak 2006] Joshua Podolak, Philip Shilane, Aleksey Golovinskiy, Szymon Rusinkiewicz and Thomas Funkhouser. *A planar-reflective symmetry transform for 3D shapes.* In ACM Transactions on Graphics (TOG), volume 25, pages 549–559. ACM, 2006. (Cited on page 55.)

[Pong 1989] Ting-Chuen Pong, Robert M Haralick and Linda G Shapiro. *Shape from shading using the facet model.* Pattern Recognition, vol. 22, no. 6, pages 683–695, 1989. (Cited on page 36.)

[Prados 2003] Emmanuel Prados, Olivier Faugeras*et al.* *Perspective shape from shading and viscosity solutions.* In Proceedings of the 9th International Conference on Computer Vision, volume 2, pages 826–831, 2003. (Cited on page 36.)

[Pressigout 2007] M. Pressigout and E. Marchand. *Real-Time Hybrid Tracking using Edge and Texture Information.* Int. Journal of Robotics Research, IJRR, vol. 26, no. 7, pages 689–713, July 2007. (Cited on page 43.)

[Prusinkiewicz 1990] Przemyslaw Prusinkiewicz and Aristid Lindenmayer. The algorithmic beauty of plants. Springer Verlag, 1990. (Cited on pages 17 and 105.)

[Prusinkiewicz 2001] Przemyslaw Prusinkiewicz, Lars Mündermann, Radoslaw Karwowski and Brendan Lane. *The use of positional information in the modeling of plants.* Proceedings of SIGGRAPH 2001, pages 289–300, 2001. (Cited on pages 17 and 94.)

[Quan 2006] Long Quan, Ping Tan, Gang Zeng, Lu Yuan, Jingdong Wang and Sing Bing Kang. *Image-based plant modeling.* ACM Trans. Graph., pages 599–604, 2006. (Cited on page 19.)

[Reche-Martinez 2004] Alex Reche-Martinez, Ignacio Martin and George Drettakis. *Volumetric reconstruction and interactive rendering of trees from photographs.* ACM Trans. Graph., pages 720–727, 2004. (Cited on page 19.)

[Remolar 2002] Inmaculada Remolar, Miguel Chover, Oscar Belmonte, José Ribelles and Cristina Rebollo. *Geometric Simplification of Foliage.* In Eurographics'02 Short Presentations, pages 397–404, 2002. (Cited on pages 17 and 93.)

[Richardson 2003] Iain E. Richardson. H.264 and mpeg-4 video compression: Video coding for next generation multimedia. Wiley, 2003. (Cited on page 108.)

[Rossignac 1999] Jarek Rossignac. *Edgebreaker: Connectivity compression for triangle meshes.* Visualization and Computer Graphics, IEEE Transactions on, vol. 5, no. 1, pages 47–61, 1999. (Cited on page 91.)

[Rothganger 2006] F. Rothganger, S. Lazebnik, C. Schmid and J. Ponce. *3D Object Modeling and Recognition Using Local Affine-Invariant Image Descriptors and Multi-View Spatial Constraints.* International Journal of Computer Vision, vol. 66, no. 3, pages 231–259, March 2006. (Cited on page 43.)

[Roudet 2011] Céline Roudet, Frédéric Payan*et al.* *Remaillage semi-régulier pour les maillages surfaciques triangulaires: un état de l'art.* Revue Electronique Francophone d'Informatique Graphique, vol. 5, no. 1, pages 27–40, 2011. (Cited on page 92.)

[Runions 2007] Adam Runions, Brendan Lane and Przemyslaw Prusinkiewicz. *Modeling Trees with a Space Colonization Algorithm.* In Eurographics Workshop on Natural Phenomena, 2007. (Cited on pages 17 and 19.)

[Rusinkiewicz 2000] Szymon Rusinkiewicz and Marc Levoy. *QSplat: A Multiresolution Point Rendering System for Large Meshes.* In Kurt Akeley, editeur, SIGGRAPH 2000, pages 343–352, 2000. (Cited on pages 44 and 91.)

[Rusinkiewicz 2004] Szymon Rusinkiewicz. *Estimating curvatures and their derivatives on triangle meshes.* In 3D Data Processing, Visualization and Transmission, 2004. 3DPVT 2004. Proceedings. 2nd International Symposium on, pages 486–493. IEEE, 2004. (Cited on page 10.)

[Russell 2013] B. C. Russell, R. Martin-Brualla, D. J. Butler, S. M. Seitz and L. Zettlemoyer. *3D Wikipedia: Using Online Text to Automatically Label and Navigate Reconstructed Geometry.* ACM Transactions on Graphics (SIGGRAPH Asia 2013), vol. 32, no. 6, November 2013. (Cited on page 134.)

[Sabin 2010] Malcolm Arthur Sabin. Analysis and design of univariate subdivision schemes, volume 6. Springer, 2010. (Cited on page 10.)

[Saito 1994] Hideo Saito and Nobuhiro Tsunashima. *Estimation of 3-D parametric models from shading image using genetic algorithms.* In Pattern Recognition, 1994. Vol. 1-Conference A: Computer Vision & Image Processing., Proceedings of the 12th IAPR International Conference on, volume 1, pages 668–670. IEEE, 1994. (Cited on page 36.)

[Schaufler 1998] Gernot Schaufler. *Image-based object representation by layered impostors.* In Proceedings of VRST '98, pages 99–104, 1998. (Cited on pages 11 and 131.)

[Schmid 2000] Cordelia Schmid, Roger Mohr and Christian Bauckhage. *Evaluation of Interest Point Detectors.* International Journal of Computer Vision, vol. 37, no. 2, pages 151–172, 2000. (Cited on page 47.)

[Schmidt 2005] Ryan Schmidt and Brian Wyvill. *Implicit sweep surfaces.* Department of Computer Science. University of Calgary, 2005. (Cited on page 133.)

[Schoenberg 1969] IJ Schoenberg. *Cardinal interpolation and spline functions.* Journal of Approximation theory, vol. 2, no. 2, pages 167–206, 1969. (Cited on page 3.)

[Seitz 2006] Steven M Seitz, Brian Curless, James Diebel, Daniel Scharstein and Richard Szeliski. *A comparison and evaluation of multi-view stereo reconstruction algorithms.* In Computer vision and pattern recognition, 2006 IEEE Computer Society Conference on, volume 1, pages 519–528. IEEE, 2006. (Cited on page 16.)

[Shade 1998] Jonathan Shade, Steven Gortler, Li-wei He and Richard Szeliski. *Layered depth images.* Proceedings of SIGGRAPH '98, pages 231–242, 1998. (Cited on pages 11 and 130.)

[Shi 2009] Shu Shi, Won J. Jeon, Klara Nahrstedt and Roy H. Campbell. *Real-time remote rendering of 3D video for mobile devices.* Proceedings of ACM MM'09, page 391, 2009. (Cited on pages 122 and 126.)

[Shi 2010] W. Shi, Y. Lu, Z. Li, and J. Engelsma. *Scalable support for 3D graphics applications in cloud.* In IEEE CLOUD '10, pages 346–353, 2010. (Cited on pages 122 and 126.)

[Shlyakhter 2001] Ilya Shlyakhter, Max Rozenoer, Julie Dorsey and Seth Teller. *Reconstructing 3D Tree Models from Instrumented Photographs*. IEEE Comput. Graph. Appl., pages 53–61, 2001. (Cited on page 22.)

[Shum 2002] Heung-Yeung Shum, Lifeng Wang, Jin-Xiang Chai and Xin Tong. *Rendering by Manifold Hopping*. International Journal of Computer Vision, vol. 50, no. 2, pages 185–201, 2002. (Cited on page 11.)

[Simon 2002] Gilles Simon and Marie-Odile Berger. *Pose Estimation for Planar Structures*. IEEE Comput. Graph. Appl., vol. 22, no. 6, pages 46–53, 2002. (Cited on page 43.)

[Sims 1990] Karl Sims. Particle animation and rendering using data parallel computation, volume 24. ACM, 1990. (Cited on page 9.)

[Sinoquet 1997] Hervé Sinoquet, Pierre Rivet and Christophe Godin. *Assessment of the three-dimensional architecture of walnut trees using digitising*. Silva Fennica, vol. 31, no. 3, pages 265–273, 1997. (Cited on pages 95 and 104.)

[Sokolov 2005] Dmitry Sokolov and Dimitri Plemenos. *Viewpoint quality and scene understanding*. In Proceedings of the 6th International conference on Virtual Reality, Archaeology and Intelligent Cultural Heritage, pages 67–73. Eurographics Association, 2005. (Cited on page 117.)

[Sterkin 2008] A. Sterkin. Interactive 3d streaming. Research@Intel Blog, June 2008. (Cited on page 122.)

[Struik 1961] Dirk Jan Struik. Lectures on classical differential geometry. Courier Dover Publications, 1961. (Cited on page 56.)

[Sun 1997] Changming Sun and Jamie Sherrah. *3D Symmetry Detection Using The Extended Gaussian Image*. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 19, pages 164–168, 1997. (Cited on page 54.)

[Szeliski 1991] Richard Szeliski. *Fast shape from shading*. CVGIP: Image Understanding, vol. 53, no. 2, pages 129–153, 1991. (Cited on page 36.)

[Tagliasacchi 2012] Andrea Tagliasacchi. *Skeletal Representations and Applications*. Sfu-cmpt tr 2012-55-1, Simon Fraser University, arXiv preprint arXiv:1301.6809, 2012. (Cited on page 22.)

[Talton 2011] Jerry O. Talton, Yu Lou, Steve Lesser, Jared Duke, Radomír Měch and Vladlen Koltun. *Metropolis procedural modeling*. ACM Trans. Graph., vol. 30, no. 2, pages 11:1–11:14, April 2011. (Cited on page 19.)

[Tan 2007] Ping Tan, Gang Zeng, Jingdong Wang, Sing Bing Kang and Long Quan. *Image-based tree modeling*. ACM Trans. Graph., page 87, 2007. (Cited on page 19.)

[Tan 2008] Ping Tan, Tian Fang, Jianxiong Xiao, Peng Zhao and Long Quan. *Single image tree modeling*. SIGGRAPH, pages 1–7, 2008. (Cited on page 20.)

[Tangelder 2008] Johan WH Tangelder and Remco C Veltkamp. *A survey of content based 3D shape retrieval methods*. Multimedia tools and applications, vol. 39, no. 3, pages 441–471, 2008. (Cited on page 85.)

[Tankus 2005] Ariel Tankus, Nir Sochen and Yehezkel Yeshurun. *Shape-from-shading under perspective projection.* International Journal of Computer Vision, vol. 63, no. 1, pages 21–43, 2005. (Cited on page 36.)

[Tauber 2004] C Tauber, H Batatia, G Morin and A Ayache. *Robust b-spline snakes for ultrasound image segmentation.* In Computers in Cardiology, 2004, pages 325–328. IEEE, 2004. (Cited on pages 9 and 13.)

[Taubin 1999] Gabriel Taubin. *3D Geometry Compression and Progressive Transmission*, 1999. (Cited on page 91.)

[Terzopoulos 1986] Demetri Terzopoulos. *Image analysis using multigrid relaxation methods.* Pattern Analysis and Machine Intelligence, IEEE Transactions on, no. 2, pages 129–139, 1986. (Cited on page 36.)

[Theobalt 2004] Christian Theobalt, Gernot Ziegler, Marcus Magnor and Hans-Peter Seidel. *Model-based free-viewpoint video: Acquisition, rendering, and encoding.* In Proceedings of Picture Coding Symposium, San Francisco, USA, pages 1–6, 2004. (Cited on page 131.)

[Thorne 2007] Matthew Thorne, David Burke and Michiel van de Panne. *Motion doodles: an interface for sketching character motion.* In ACM SIGGRAPH 2007 courses, page 24. ACM, 2007. (Cited on page 39.)

[Tisseron 1988] Claude Tisseron. Géométries affine, projective et euclidienne. Hermann, 1988. (Cited on page 59.)

[Touma 1998] Costa Touma and Craig Gotsman. *Triangle mesh compression.* In Graphics interface, volume 98, pages 26–34, 1998. (Cited on page 91.)

[Tsai 1994] Ping-Sing Tsai and Mubarak Shah. *Shape from shading using linear approximation.* Image and Vision Computing, vol. 12, no. 8, pages 487–498, 1994. (Cited on page 37.)

[Vacchetti 2004] L. Vacchetti, V. Lepetit and P. Fua. *Stable Real-Time 3D Tracking Using Online and Offline Information.* IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 26, no. 10, pages 1391–1391, 2004. (Cited on pages 43 and 46.)

[Vaillant 2013] Rodolphe Vaillant, Loïc Barthe, Gaël Guennebaud, Marie-Paule Cani, Damien Rohmer, Brian Wyvill, Olivier Gourmel, Mathias Paulin, LJK-Grenoble Universités-Inria and CPE Lyon-Inria. *Implicit skinning: real-time skin deformation with contact modeling.* ACM Transactions on Graphics (TOG), vol. 32, no. 4, page 125, 2013. (Cited on page 133.)

[Vázquez 2001] Pere-Pau Vázquez, Miquel Feixas, Mateu Sbert and Wolfgang Heidrich. *Viewpoint Selection using Viewpoint Entropy.* In VMV, volume 1, pages 273–280, 2001. (Cited on page 117.)

[Vázquez 2009] Pere-Pau Vázquez. *Automatic view selection through depth-based view stability analysis.* The Visual Computer, vol. 25, no. 5-7, pages 441–449, 2009. (Cited on page 117.)

[Wang 2006] Rui Wang, Wei Hua, Zilong Dong, Qunsheng Peng and Hujun Bao. *Synthesizing trees by plantons.* Vis. Comput., vol. 22, no. 4, pages 238–248, April 2006. (Cited on page 19.)

[Wang 2010] Aobo Wang, Cong Duy Vu Hoang and Min-Yen Kan. *Perspectives on Crowdsourcing Annotations for Natural Language Processing*, 2010. (Cited on page 69.)

[Warren 2001] Joe Warren and Henrik Weimer. Subdivision methods for geometric design: A constructive approach. Morgan Kaufmann, 2001. (Cited on page 10.)

[Weber 1995] Jason Weber and Joseph Penn. *Creation and Rendering of Realistic Trees.* SIGGRAPH 1995, pages 119–128, August 1995. (Cited on pages 17, 94 and 107.)

[Wiles 2001] Charles S. Wiles, Atsuto Maki and Natsuko Matsuda. *Hyperpatches for 3D Model Acquisition and Tracking.* IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 23, no. 12, pages 1391–1403, 2001. (Cited on page 43.)

[Wilson 2003] Andrew Wilson and Dinesh Manocha. *Simplifying complex environments using incremental textured depth meshes.* In Proceedings of SIGGRAPH '03, pages 678–688, 2003. (Cited on page 131.)

[Wither 2009] J. Wither, F. Boudon, M.-P. Cani and C. Godin. *Structure from silhouettes: a new paradigm for fast sketch-based design of trees.* In Computer Graphic Forum. Special issue: Eurographics 2009, volume 28 (2), pages 541–550, 2009. (Cited on page 19.)

[Woodford 2005] Oliver Woodford and Andrew W Fitzgibbon. *Fast image-based rendering using hierarchical image-based priors.* In Proc. BMVC, volume 1, pages 260–269, 2005. (Cited on page 11.)

[Xiao 2004] Jing Xiao, Simon Baker, Iain Matthews and Takeo Kanade. *Real-Time Combined 2D+3D Active Appearance Models.* In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, volume 2, pages 535–542, June 2004. (Cited on page 43.)

[Yuille 2006] Alan L Yuille and Daniel Kersten. *Vision as Bayesian Inference: Analysis by Synthesis?* Trends in Cognitive Sciences In Probabilistic models of cognition, 2006. (Cited on page 34.)

[Zabrodsky 1995] H. Zabrodsky, S. Peleg and D. Avnir. *Symmetry as a Continuous Feature.* IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 17, pages 1154–1166, 1995. (Cited on page 54.)

[Zeng 2006] Jiguo Zeng, Yan Zhang and Shouyi Zhan. *3D Tree Models Reconstruction from a Single Image.* ISDA, pages 445–450, 2006. (Cited on pages 20 and 27.)

[Zhang 2004] C Zhang. *A survey on image-based rendering—representation, sampling and compression.* Signal Processing: Image Communication, vol. 19, no. 1, pages 1–28, January 2004. (Cited on page 11.)

[Zhao 2012] Yuxiang Zhao and Qinghua Zhu. *Evaluation on crowdsourcing research: Current status and future direction.* Information Systems Frontiers, pages 1–18, 2012. (Cited on page 69.)

[Zhao 2013] Shanghong Zhao, Wei Tsang Ooi, Axel Carlier, Géraldine Morin and Vincent Charvillat. *3D mesh preview streaming.* In MMSys, pages 178–189, 2013. (Cited on pages 11, 90 and 121.)

[Zhu 2011] Minhui Zhu, Sebastien Mondet, Géraldine Morin, Wei Tsang Ooi and Wei Cheng. *Towards peer-assisted rendering in networked virtual environments.* In Proceedings of the 19th ACM international conference on Multimedia, pages 183–192, 2011. (Cited on pages 12, 90 and 123.)

[Zitova 2003] Barbara Zitova and Jan Flusser. *Image registration methods: a survey.* Image and vision computing, vol. 21, no. 11, pages 977–1000, 2003. (Cited on page 85.)

[Zwicker 2001] Matthias Zwicker, Hanspeter Pfister, Jeroen van Baar and Markus Gross. *Surface Splatting.* In Eugene Fiume, editeur, SIGGRAPH 2001, Computer Graphics Proceedings, pages 371–378. ACM Press / ACM SIGGRAPH, 2001. (Cited on pages 44 and 45.)

[Zwicker 2004] Matthias Zwicker, Jussi Räsänen, Mario Botsch, Carsten Dachsbacher and Mark Pauly. *Perspective accurate splatting.* In GI '04: Proceedings of Graphics Interface 2004, pages 247–254, School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, 2004. Canadian Human-Computer Communications Society. (Cited on page 45.)