Chapter 1

Introduction and motivation

1.1 Motivation: Real-time rendering of large and complex scenes

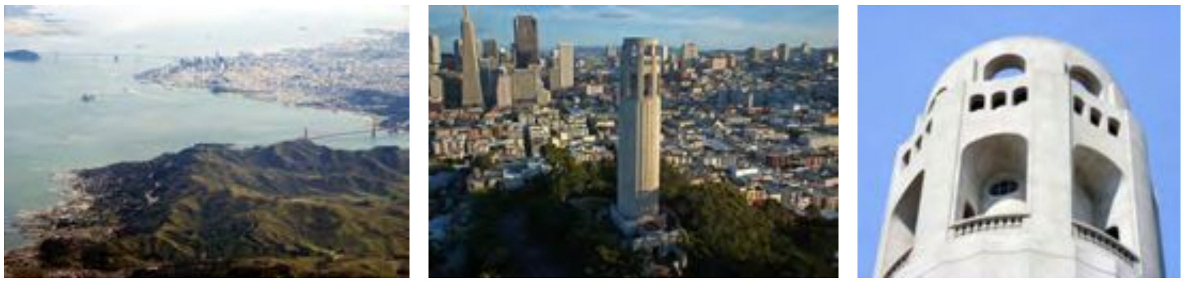
Figure 1.1. Illustration of a multiscale scene. *Sources: (Left-to-right) San-Francisco by arcortvriend , San-Francisco from www.staysf.com, Coit-tower from http://wsisf.com*

Photorealism has always been a major goal for computer graphics (CG). Today, one of the key problems is the management of details. Rendering large multi-scale scenes (such as the one presented in figure 1.1) and very detailed objects (Figs. 1.2 and 1.3) accurately and efficiently is now one of the most challenging problems in computer graphics, not only for real-time applications, but also for CG featured films. Usual rendering methods tend to be very inefficient for highly complex scenes because rendering cost is proportional to the number of primitives. In such scenes, many geometrical details have to be accounted for in each single pixel. The small size of primitives creates high-frequency signals that lead to aliasing artifacts which are costly to avoid. In order to produce high quality rendering, all lighting contributions of these details must be integrated in order to compute the final color of a given pixel. This integration poses two kinds of problems: how to do it efficiently (in terms of performance), and how to do it accurately (in terms of quality).

In this section, we will show how current rendering approaches reach their limits both in terms of quality and performance when dealing with very complex and multi-scale scenes. In this context, scalability is an absolute necessity. We will see that massive per-pixel supersampling is not affordable for real-time applications, and does not scale to an increasing number of details to be integrated per-pixel. Thus, geometrical simplification approaches have to be used, in order to deal with large amounts of geometrical data, and to prevent aliasing problems. But such simplifications inevitably remove details that would have contributed to the final image, and fails to preserve important shading effects (eg. roughness). The challenge in this context is to be able to keep shading details while allowing interactive rendering and maintaining a reasonable memory consumption. We will demonstrate how some form of geometrical pre-filtering is required to tackle this problem, and explain how we propose to rely on a volumetric enriched representation to handle it.

1.1.1 Limits of current mesh-based rendering approaches

All current rendering models used today to display geometry are based on a triangle or quad mesh representation. With this representation, objects are modeled by their surface, and are decomposed into simple primitives for rendering. Rendering is a view-based sampling operation. The idea is to sample a 3D scene in order to reconstruct a 2D image. The 3D scene represents the continuous source signal, while the 2D image is the signal we want to reconstruct. According to the Nyquist-Shannon sampling theorem [Nyq28], sampling frequency needs to be at least twice the highest signal frequency in order to reconstruct it correctly. Sampling below this Nyquist limit leads to so-called "aliasing". Conceptually, this means that we need no less than one sample for each peak and another for each valley of the original signal. That means that for each surface primitive to render, at least two samples need to be calculated in order to accurately reconstruct its contribution to the final image. Thus, increasing the geometrical complexity of a scene necessarily means increasing the sampling density used to compute the rendered image in the same proportions, in order to ensure a high quality reconstruction.

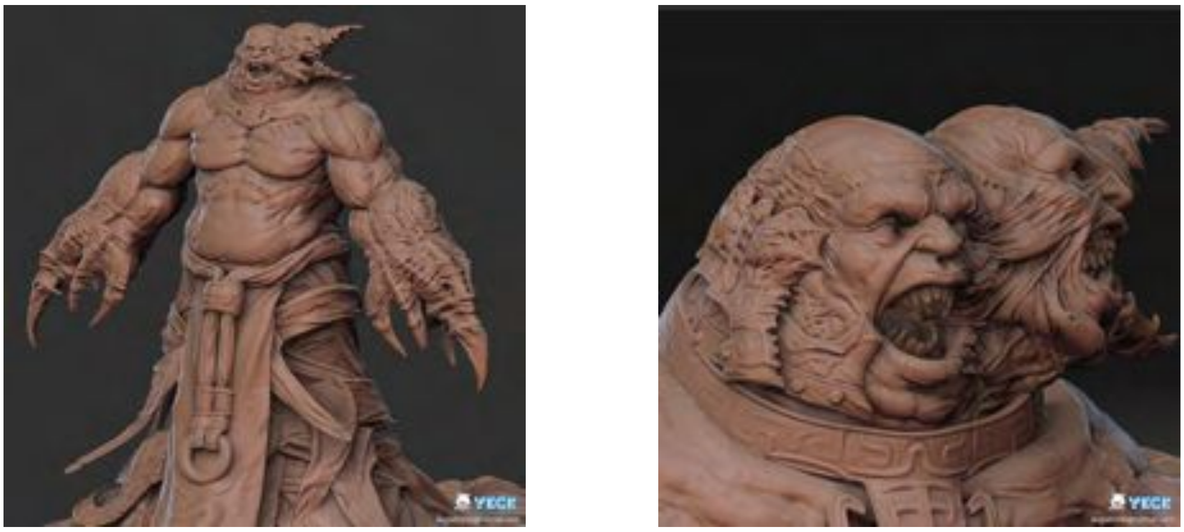

Figure 1.2. Example of an highly detailed mesh modelized using ZBrush [Spe08]. Image courtesy of Yeck.

The problem is that the amount of available processing power does not scale as fast as the need for more and more complex objects, and even if it did, increasing the computation proportionally to the complexity of a scene do not seems to be a scalable or sustainable solution. Similarly, the amount of memory available for rendering is a limited resource and storing and quickly accessing arbitrary large and complex scenes is a challenging problem.

The primary challenge in computer graphics has always been to bit these constraints in order to produce a more and more detailed rendering, without increasing the processing time and needed storage in the same proportions. Real-time rendering of highly complex geometrical scenes poses two kinds of problems. First, a quality problem: how to accurately integrate the shading contributions of all details, in order to prevent aliasing and capture expected lighting effects. Second, a performance problem: how to efficiently deal with large amounts of geometrical data, both in terms of computation and storage.

Classical mesh+texture representation

The classical way to deal with detail rendering is to split geometry representation into a coarse triangle-based surface mesh representation and fine scale surface details, reflectance and illumination properties, specified with 2D texture maps [Cat74] and used as parameters in a local illumination model. When objects can be seen from a relatively limited range of distances, this representation makes it possible to use a unique coarse mesh and to pre-filter detail maps linearly and separately. The idea is to estimate the averaged outgoing radiance from a surface to a pixel by applying the local illumination model on the averaged surface parameters from the map, instead of averaging the result of this application on fine-grained details. To do so, the classical approach is to rely on MIP-mapping [Wil83], with multiple map resolutions pre-computed before rendering. Such texture filtering does not prevent aliasing to appear at the boundary of meshes and on the silhouette of objects. This issue is usually addressed using a multi-sampling technique [Ake93], estimating per-pixel triangle visibility at higher frequency (>1 sample per pixel) than the shading itself.

This classical approach works well for moderate range of view distances, and low-detail meshes, when triangles cover at least one pixel on the screen and silhouettes do not contain thin details. But when highly detailed meshes viewed from a wide range of distances need to be rendered, many mesh triangles project to the same screen pixel and simply relying on the pre-filtering of the maps and a limited multisampling becomes insufficient to integrate all surface details, and leads to aliasing artifacts.

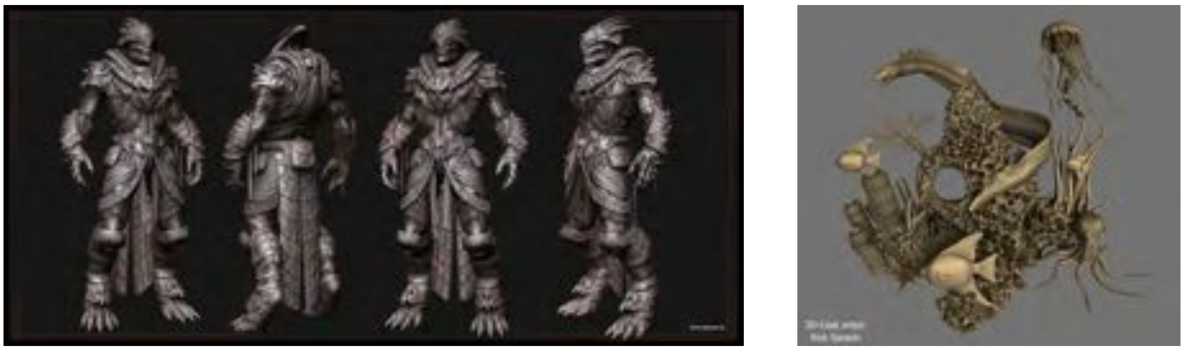

Figure 1.3. *Left:* Example of an highly detailed character mesh modelized using Z-Brush. *Right:* Very complex mesh modelized using 3D-Coat, a voxel-based 3D sculpting tool. Images courtesy of Benerdt.de (left) and Rick Sarasin (right)

Adaptive supersampling

The classical way to integrate per-pixel geometrical details is to rely on supersampling [Whi80, CPC84]. For each pixel, a weighted average of the illumination coming from all meshes elements and falling into that pixel is computed, i.e., an integral of the contributions of all these elements, which can be extremely costly. The outgoing radiance from a surface is given by a local illumination model as a function of the incident radiance and the surface properties. Computing this integral numerically during rendering (using massive sampling) can be extremely costly and many adaptive and stochastic multi-sampling methods have been proposed to speed-up its computation by optimizing the placement and distribution of samples. Adaptive supersampling allows us to push back slightly the limit of complexity that can be handled per-pixel, but still does not scale with an arbitrary increase of geometrical complexity. In addition, the cost of such supersampling of the geometry is usually not affordable for real-time application.

Mesh simplification and geometric LOD

In such situations, the approach usually employed to deal with aliasing problems and reduce rendering time is to simplify the geometry itself, in order to adapt it to the rendering resolution. This is done by relying on geometric level-of-details (LOD) approaches that progressively remove mesh details. The idea is to maintain a constant number of primitives to be rendered per screen pixel, whatever the viewing distance and complexity of the original mesh.