HAL
open science

Flexible Point-Based Rendering on Mobile Devices

Florent Duguet, George Drettakis

To cite this version:

Florent Duguet, George Drettakis. Flexible Point-Based Rendering on Mobile Devices. RR-4833, INRIA. 2003. inria-00071753

HAL Id: inria-00071753

https://inria.hal.science/inria-00071753v1

Submitted on 23 May 2006

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Flexible Point-Based Rendering on Mobile Devices

Florent Duguet — George Drettakis

N° 4833

Mai 2003

THÈME 3

Rapport de recherche

ISSN 0249-6399 ISRN INRIA/RR--4833--FR+ENG

Flexible Point-Based Rendering on Mobile Devices

Florent Duguet*, George Drettakis †

Thème 3 — Interaction homme-machine,
images, données, connaissances
Projet REVES

Rapport de recherche n° 4833 — Mai 2003 — 20 pages

Abstract: Point-based rendering is a compact and efficient means of displaying complex geometry. For mobile devices which typically have limited CPU or floating point speed, limited memory, no graphics hardware and a small display, a hierarchical packed point based representation of objects is particularly well adapted. We introduce ρ-grids, which are a generalization of previous octree based representations and analyse their memory and rendering efficiency. By storing intermediate node attributes, our structure allows flexible rendering, permitting efficient local image refinement, required for example when zooming into very complex scenes. We also introduce a novel and efficient one-pass shadow mapping algorithm using this data structure. We show an implementation of our method on a PDA, which can render objects sampled by 1.3 million points at 2.1 frames per second;the model was originally made up of 4.7 million polygons.

Key-words: point-based rendering, shadow maps

* http://www-sop.inria.fr/reves/personnel/Florent.Duguet
† http://www-sop.inria.fr/reves/personnel/George.Drettakis

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles, BP 93, 06902 Sophia Antipolis Cedex (France)
Téléphone : +33 4 92 38 77 77 — Télécopie : +33 4 92 38 77 65

Rendu Flexible à Base de Points sur Plateformes Mobiles

Résumé : Le rendu à base de points est une méthode compacte et efficace pour l'affichage de géométries complexes. Les plateformes mobiles ont typiquement un seul processeur et des capacités de calculs en virgule flottante limitées, peu de mémoire, un affichage réduit et aucun processeur dédié au graphique. Dans ce cadre, une représentation hiérarchique compacte à base de points pour les objets graphiques est particulièrement bien adaptée. Nous introduisons les ρ-grids, qui sont une généralisation des représentations à base d'octrees existantes, et nous analysons leur efficacité sur le plan de la mémoire et du rendu. En stockant les attributs intermédiaires dans la hiérarchie (couleur, normale), notre structure permet un rendu flexible, avec un raffinement local de l'image, ce qui est requis par exemple en examinant des objets dans une scène complexe. Nous introduisons aussi un nouvel algorithme efficace de calcul d'ombres basé sur la technique des cartes d'ombres, utilisant notre structure et ne nécessitant qu'une passe de rendu. Nous présentons une implémentation de notre méthode sur un PDA, pouvant rendre des objets échantillonnés à 1,3 millions de points à 2,1 images par secondes. Le modèle est originellement composé de 4,7 millions de polygones.

Mots-clés : rendu par points, cartes d'ombres

1 Introduction

Recent years have seen the proliferation of complex virtual environments with the deployment of the more ubiquitous computer, which we now use at work, in the household, for leisure etc. With the increasing demand for detailed and high quality models, typical scene size easily reaches millions of primitives. At the same time, display devices are becoming more diverse, including mobile or embedded platforms such as PDAs or mobile phones, thus significantly increasing the user base for complex virtual environments.

The main motivation of our work is to allow interactive display of complex scenes on different kinds of platforms with heterogeneous rendering capabilities, and more precisely on mobile devices. Even though embedded systems have decent computational capabilities in terms of basic CPU operations, their memory is quite limited, they typically lack floating point units, and their display is limited to around a hundred thousand pixels (resolutions of 240x320 are typical). These elements change a number of our basic assumptions about rendering tradeoffs.

For instance, the geometric complexity of scenes we want to render, often measured in polygons, can be tens of thousands or even millions, whereas the number of pixels e.g., on a PDA may be less than one hundred thousand. Using standard rasterization operations on polygonal geometry would be a waste. To accommodate all the above constraints, i.e., limited memory and floating point speed as well as limited display size, we choose a packed hierarchical point-based representation for rendering.

Hierarchical techniques have already been used for point-based rendering using very efficient algorithms for both data storage[2] and rendering[15]. This work has inspired our choices of data structures and rendering algorithms. However, we introduce a more flexible structure, which allows locally adaptive progressive rendering. This is achieved by explicitly storing intermediate attributes, as in [16, 15].

We generalize the octree structure presented by Botsch et al.[2], to a class of hierarchical structures we call $\rho-$grids. In this context, i.e., the storage of intermediate attributes, we show that the choice of a 3x3x3 subdivision scheme, which we call a tri-grid, is more efficient in storage and rendering speed than an octree.

We introduce a multi-level rendering algorithm which stops the traversal of the hierarchy at the appropriate level, and directly renders using the stored intermediate sample attributes. For devices with limited resources, actually traversing the hierarchy is a very costly operation. Our structure allows us to define an appropriate ordering for rendering samples, resulting in a fast one-pass shadow map algorithm.

In this work we analyze the storage and rendering time complexity of our structure. We show that our choice of data structure is based on a careful decision on the tradeoff between the cost of traversing the hierarchy and the cost of processing a node (rendering in our case).

In the next section, we briefly present previous related work. We discuss the method of Botsch et al.[2] in some detail, since we have been inspired by some of the choices made there. In section 4 we introduce our general framework for packed hierarchical structure, and present a comparative study for ρ-grids. In section 5 we describe our rendering algorithm.

We present results and a discussion of implementation issues in section 6, and conclude (section 7).

2 Previous Work

For very complex scenes, the traditional graphics pipeline can be very wasteful, with much effort spent transforming and rasterizing large numbers of geometric primitives which may cover less than a pixel. Point based rendering, introduced as early as 1985 by Levoy and Whitted [12], addresses this issue by point-sampling complex geometry [18, 17], and rendering an appropriate number of points depending on the screen size of the complex object.

Several papers have been published in this domain over the last few years. Grossman and Dally [9] generated point representations of objects in a preprocess, and presented efficient rendering algorithms of these point sets. Q-splat is an approach permitting the visualisation of very complex models[15], in particular those that do not fit in main memory. This

Figure 1: (Left) Our structure used for hierarchical point-based rendering of a 4.7M polygon model at 2.1 fps on a 200Mhz IPAQ, sampled at 1.3 M points total. The multi-level approach restricts the number of points rendered depending on the view. Notice the shadows in the scene. (Right) The dragon model at 2.3 fps.

approach was extended to handle network transmission[16]. Surfels[14], surface splatting[19] and point-set surfaces[1] address, among other aspects, high-quality rendering issues. Other approaches perform combined rendering of points and polygons[5, 4] or even lines[6].

The question of how to point-sample complex objects has been addressed by Pfister et al.[14], who create an octree representation. Inspired by techniques related to geometric coding[7], a more compact representation is presented by Botsch et al.[2], in which an octree is used to code point positions implicitly. This approach is extremely efficient in memory, and the authors present a very rapid rendering algorithm. This work has inspired our approach, for our choices for both storage and rendering. We show how the octree structure used by Botsch et al.[2] can be generalised to different kinds of subdivisions. We will also extend their approach by storing intermediate attribute samples so that rendering can be flexible and thus more efficient.

Our approach combines the advantages of hierarchical rendering of Q-splat[15, 16] and the compact and efficient storage of Botsch et al.[2].

3 Overview

Our goal is to render complex scenes, containing hundreds of thousands or even millions of primitives on mobile platforms. To achieve this goal we choose a point sample representation coded in a hierarchical structure. We need this structure to be flexible, in terms of storage and rendering. To develop such a structure, we introduce the class of *recursive grids*, which we call ρ-grids. These are recursive grids with a regular and uniform subdivision at each level. ρ-grids (with a subdivision of $\rho x \rho x \rho$ for each cell) can be seen as a generalization of the octree (subdivided into $2x2x2$ for each cell).

We first explain how we implicitly code point position, as in [2], and how we code additional attributes such as color and normal information. To achieve flexible multi-level display, we sample and store such attributes at intermediate levels of the hierarchy. If a whole branch of the tree projects to a single pixel of the display, it can be represented by a single sample. We analyze the memory consumption of this class of hierarchical structures and demonstrate that the tri-grid is most efficient if intermediate attributes are stored, for the range of bit-code sizes we use.

Our rendering algorithm is similar to [2], and is based on the efficient projection of samples onto the screen. We introduce an efficient multi-level rendering approach which directly renders intermediate nodes and uses frustum culling. We also introduce an efficient one-pass shadow map algorithm using this structure. We analyze rendering times and show that the tri-grid is the best overall choice when considering the tradeoff between rendering and storage costs, for our range of sample attributes sizes.

Such compact and efficient structures become particularly important for mobile platforms. We have implemented our structure on a 200Mhz Compaq iPAQ H3850 running PocketPC 2002, and we describe implementation and system issues which were involved in this effort.

4 Framework for structured point-sampled geometry

Before presenting the general recursive ρ-grid framework, we will discuss our sampling strategy. We then analyze memory consumption of this class of structures.

4.1 Sampling

Our sampling strategy is simple and general: for each cell of the hierarchy, whether intermediate or leaf, we sample the geometry at the point of the object closest to the center of the cell. If the cell contains no object, the cell is flagged as empty. We can thus sample any kind of object, as long as the following two operations are available:

- *IntersectAxisAlignedBox*: returns true if the object intersects, or is contained in, a given axis aligned box.
- *GetSampleAt*: returns a sample and its attributes for a given input point (cell center), and geometry primitive.

As discussed below, sample position is coded implicitly with the hierarchical data-structure. In addition we can code normal and material indices. In what follows we use a 16-bit code for a 13-bit quantized normal index (as in [2]) and for indexing 8 materials. Evidently, we need more bits if we have more materials, or colors; these codes vary from 32 to 48 bits.

In our recursive grid structure, we store what we call *intermediate sample attributes*. By this we mean normal and potentially material/color codes for interior (i.e., non-leaf) nodes of the hierarchy. As discussed later, these intermediate samples allow us to effect efficient and flexible multi-level rendering with limited computational and graphics processing resources, such as on PDA's.

Evidently, more involved sampling strategies could be employed, for example by using a bottom-up filtering approach, similar in a way to texel approaches[13].

4.2 Recursive grids

Recursive grids were first introduced as acceleration structures for ray-tracing. The main benefit of such structures is both the flexibility given by the hierarchical structure, and the optimized algorithms for grid traversal at each level of the hierarchy [8, 11]. We will describe here another usage of recursive grids, i.e., as a compact point-sampling structure.

An octree can be seen as the simplest recursive grid: each cell is subdivided into eight subcells. We will focus on regular recursive grids, that is grids for which the number of subcells is ρ^3, with ρ the number of subcells per dimension, and with uniform subdivision. The octree is this structure for $\rho = 2$. We call **tri-grid**, the recursive grid for $\rho = 3$.

As discussed previously, point based rendering samples can be structured in a hierarchy which implicitly encodes their positions[2]. Sample positions are aligned with octree cell centers, so that position does not need to be stored. Botsch et al.[2] describe a very compact

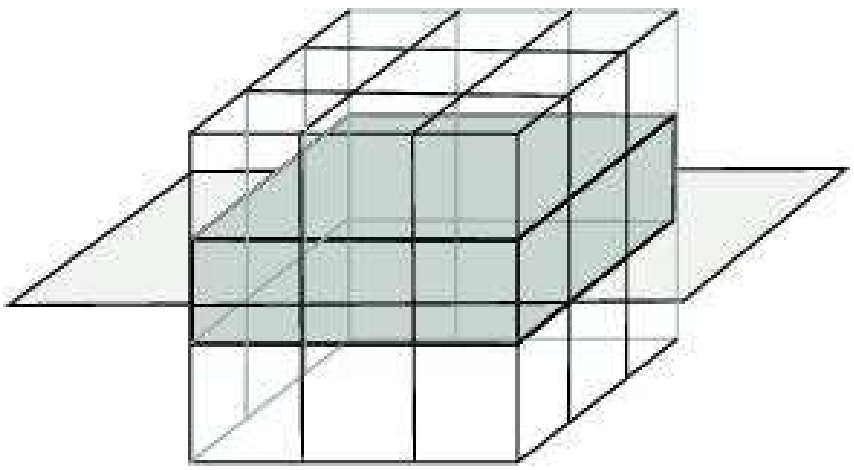


Figure 2: Illustration of dimension for a plane.

representation of the octree, minimizing storage cost. We extend these ideas to general ρ-grids, and analyze storage cost and rendering time, depending on various parameters. The hierarchy of a ρ-grid can be efficiently encoded using ρ^3 bits per intermediate cell. We use the approach of Botsch et al.[2], in which 8 bits are used to code the subcells of an octree. A bit is set if the sub-cell is non-empty. By extension the tri-grid requires 27 bits to code position.

In addition to position however, we will consider coding of other attributes, notably normals and colors.

4.3 Memory consumption for $\rho-$grids

We define α to be the number of non-empty sub-cells for a given cell. We denote α_ρ the average of α over all cells. For the tri-grid, and for a depth of subdivision 5 or greater, we have observed that the value of α_ρ is approximately 9, independent of the type of model (see models and statistics described in Table 4). We define $d = \frac{log\alpha_\rho}{log\rho}$. Thus for the tri-grid, and assuming an α_ρ of 9, $d = 2$. Thus, intuitively, d is related to dimension of the models, since all the models used are surfaces of dimension two. Another way to illustrate this is to consider a horizontal plane (see Fig. 2) in a tri-grid, in which α_ρ is exactly 9, and d is 2.

4.3.1 Structure Size

We base our analysis of memory consumption on n, the number of samples in the finest representation of the model, that is the number of leaf cells. We will compute the cost of

the hierarchical structure accounting for both position and additional attributes needed to represent intermediate levels (for multi-resolution rendering capabilities).

We note m the maximum depth of the ρ-grid, and N_i the number of cells at depth i. We have $N_m = n$, and $N_i = \alpha_\rho N_{i-1}$. The number of intermediate cells is the sum over all levels but last, which is:

$$N = \sum_{i=0}^{m-1} N_i = \frac{n-1}{\rho^d - 1}$$

The cost of the structure by sample is given dividing by n, and counting ρ^3 bits per cell:

$$S_c = \frac{\rho^3}{\rho^d - 1}$$

This size increases with ρ indicating the octree is the optimal structure in storage, if we only consider the coding of position as discussed in [2].

4.3.2 Intermediate samples

If we also store intermediate sample attributes however, this analysis needs to be generalized. The number of intermediate samples is given by the number of intermediate cells. We note σ the size, in bits, of a sample attribute. The memory cost of the intermediate samples attributes, per leaf sample, is thus given by:

$$S_s = \frac{\sigma}{\rho^d - 1}$$

The total cost of the hierarchical structure is thus:

$$S = \frac{\sigma + \rho^3}{\rho^d - 1}$$

In Table 1 we show how memory consumption of our structures varies with σ and with ρ. As we can see, the tri-grid has the lowest memory consumption for attributes coded with between 4 and 16 bits. For a higher number of bits, 4- or 5− grids may be competitive, but we show later that they are not as efficient for rendering.

5 Rendering

In the context of rendering on mobile devices, rasterization has a significant cost. But even if rasterization were free, the projection of vertices in screen coordinates cost a 4x4 matrix multiplication. This operation has to be done for each vertex of the model. Projecting a vertex costs at least 16 multiplications and 12 additions. This underlines the fact that polygonal representation of complex objects is unsuitable in the context of mobile devices. An unstructured point based representation could be used, but the projection problem would

σ	$\rho = 2$	$\rho = 3$	$\rho = 4$	$\rho = 5$
0	**2.66**	3.38	4.26	5.21
4	4	**3.88**	4.53	5.375
8	5.33	**4.38**	4.8	5.54
12	6.67	**4.88**	5.06	5.71
16	8	5.38	**5.33**	5.88
32	13.33	7.38	**6.4**	6.54
48	18.67	9.38	7.47	**7.21**

Table 1: Variation of ρ-grid memory consumption per leaf sample with ρ and the number of bits per sample attribute (σ). The bold numbers are the most efficient structure for a given σ. σ=0 corresponds to no sample attribute, as in [2].

remain. Structured hiearchies of point samples are much more appropriate. In what follows, we present an efficient algorithm for rendering which is a generalization of [2] to ρ-grids. We then introduce our flexible rendering algorithm as well as a new one-pass shadow-map rendering algorithm.

5.1 Basic Rendering Algorithm

The basic rendering algorithm is a generalization of the octree approach[2] to ρ-grids. It proceeds as follows: given a ρ-grid, we project the center using a standard projection in homogeneous coordinates. We then precompute a table of displacement vectors[2]. This table is composed of ρ^3 vectors, corresponding to the relative sub-cell centers projected in image space. Since homogeneous projections are linear (matrix multiplications), the projection of a sub-cell center can be expressed given its projected parent center and the projection of a displacement vector. This table of ρ^3 vector entries is precomputed for each level (up to the precision of the model), and at each modification of the viewing position (4x4 projection matrix).

Using this approach, the center of a cell in the ρ-grid can be computed with three additions from the center of the parent cell. To get the final projection, the center has to be dehomogeneized with two divisions, or using a more efficient approach as detailed in 6.1. The table of displacements $d_{i,j,k}$ is precomputed as follows. The displacements giving the first level from the root level are computed using standard projection. This is shown below, Eq. (1):

$$\vec{d}^{(1)}_{i,j,k} = \frac{i}{\rho}\vec{e_i} + \frac{j}{\rho}\vec{e_j} + \frac{k}{\rho}\vec{e_k} \tag{1}$$

with $i, j, k \in [-\frac{\rho-1}{2}; \frac{\rho-1}{2}]$ and $\vec{e_i}$, $\vec{e_j}$, $\vec{e_k}$ the three projected unit basis vectors.

Subsequent levels are incrementally computed with a multiplication per vector, as shown next (Eq. (2)).

$$\vec{d}^{(n)}_{i,j,k} = \frac{1}{\rho}\vec{d}^{(n-1)}_{i,j,k} \tag{2}$$

Then, for each subcell i, j, k of a cell c, we compute the projected center with three additions:

$$\vec{c}^{(n)}_{i,j,k} = \vec{d}^{(n)}_{i,j,k} + \vec{c}^{(n-1)}$$

The recursive rendering algorithm is described next.

```
Render (cell, center, level)
  if cell is a leaf
    for each subcell
      if sampled
        compute position
        Draw Sample
  else
    for each subcell
      if exists
        compute subcenter
        Render (subcell, subcenter, level+1)
```

5.2 Flexible Rendering: Multi-level, Splats and Culling

The rendering algorithm described so far is efficient, but will always render the leaf nodes. This is wasteful if the projected size of intermediate hierarchy cells is smaller than a pixel, for example when zooming out of an object. Using a structure such as that described by Botsch et al.[2], one could potentially reconstruct normals and surface/color properties, for example by averaging up the attributes in the hierarchy. However, in this context of the rendering algorithm described so far, this operation is more expensive than the rendering (i.e., point projection, splats) itself. Indeed, for example, averaging *quantized* normals would be surely more expensive than actually rendering sub-samples.

To render intermediate nodes, we compute a conservative approximation of the projection of a cell (an axis aligned cube). In practice we compute the screen-space bounding rectangle of the cell. This is done by computing a displacement table in a manner similar as for cell centers. The screen-space bounds are given as (min_x, min_y, max_x, max_y); we also compute $min_{z/w}$, which is the minimum homogeneous depth of the bounding box.

For each level, if the extent of the projected cell bounding box is less than a pixel, we draw the intermediate sample. As with previous point-based approaches we render splats to represent these samples. Denote $dx = (max_x - min_x)$ the extent in x and s the splat size. To determine whether we stop at this level we need to perform two tests in homogeneous coordinates, which costs two multiplications and two tests, as shown next:

$$\frac{dx}{w} \leq s, w > 0 \tag{3}$$

which is equivalent to

$$dx \leq w * s, w > 0 \tag{4}$$

We can also perform an efficient frustum culling test using the same bounding information: we test the bounds of the projected cell against the bounds of the screen in the same manner. If the intermediate cell is outside the bounds of the screen, we ignore it and its children.

If we do not do culling, we can precompute the extent S as:

$$S = max_{x,y}(max - min) \tag{5}$$

for each level, thus avoiding all min-max computations for each cell.

5.3 Shadows

Our approach is well adapted to efficient shadow map computation. In addition, when rendering the shadow map from the light source, we can use larger splats, since it is often unimportant to capture highly detailed shadows, especially given the limited screen size of our PDA's.

If we use the standard shadow-map algorithm, for each pixel of the image (with depth), we transform to the space of the light source using a 3x3 matrix. This operation needs 9 multiplications, 6 additions, and a test in the shadow map per pixel. For a typical iPAQ screen, this step would require 690,000 multiplies, and 540,000 additions or tests.

In order to avoid expensive matrix multiplications, we perform the two passes of the shadow-map as follows: first, we render to a depth map using the projection matrix of the light source, and the associated projected displacement vectors. Each sample in light space is rendered using the efficient incremental technique of Section 5.1. As a result, we avoid the expensive matrix computation to transform the points into light space for the shadow map comparison.

Then we render the scene computing projected positions for both the light source projection matrix and camera projection matrix. For each sample, we perform the usual test against the depth map using light projected coordinates.

Using this approach requires 3 additions per intermediate or leaf cell. Assuming a multiplication is 3 times slower than an addition or a test [10], we could render 870,000 cells and samples for the same processing time. Note that using flexible rendering, we rarely reach this number of samples and cells.

A single pass method for shadow-map computations can also be used for directional light sources. Given the light direction, we can attribute a strict order in the rendering of the hierarchical structure. As show in Figure 3, we can define an ordering on subcells so that cells are rendered in front to back order[2]. This ordering is precomputed once for the 27 cells for a given light source position, by projecting subcell centers onto the direction of the light source. If we traverse the hierarchy in this order, we can guarantee that lit samples are rendered first, and samples in shadow are rendered afterwards. Thus, we compute the shadow map and the view from the camera at the same time, and perform the depth

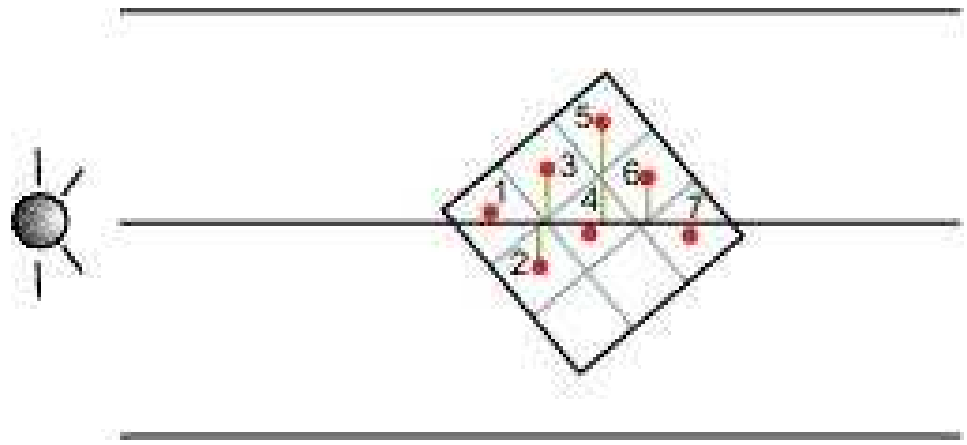


Figure 3: Ordering for shadow map rendering.

comparisons as we render. We thus avoid the first rendering pass in shadow map coordinates required by the previous algorithm. The computational gains compared to two-pass shadow mapping are shown in Table 5.

It is important to note that these algorithms (multi-level, splatting and shadows) are given for general ρ-grids and could be applied to any specific type of grid, the octree being one of them. Our choice of the tri-grid for both storage and rendering is justified by Tables 5 and 1. Different grid types could be used within the same runtime environment with a minimal overhead.

5.4 Rendering cost

In the case of the naive renderer, we traverse all samples, thus going through the entire acceleration structure. The rendering cost is given by the number of cells to be rendered and the number of operations for each cell. We have three additions per sub-cell visited which is $3\rho^d$, and a shift per sub-cell which is ρ^3. The number of intermediate cells per sample is $\frac{1}{\rho^d-1}$. Thus, number of operations per sample for naive rendering is given as follows:

$$T = \frac{3\rho^d + \rho^3}{\rho^d - 1}$$

This time calculation applies only to basic rendering, i.e., rendering all samples as points, without splats or shadows. If we use other rendering algorithms such as rendering with shadows, splats, frustum culling, the number of operations per cell grows. For example, adding shadows with our single pass algorithm leads to 6 operations per cell.

Table 2 summarizes the rendering cost in terms of operations, for different values of ρ.

ρ	2	3	4	5
basic (3)	6.6	6.7	7.5	8.4
shadows (6)	10.7	10.1	10.6	11.5

Table 2: Rendering cost (in operations) for different values of ρ

As we can see from Table 2, the tri-grid is essentially as efficient as the octree, but more efficient than the 4- or 5-grid. We thus see that the tri-grid is the best choice if one considers both rendering and storage cost (Tables 5 and 1). The trade-off between hierarchy traversal and node processing has also been studied in the context of ray tracing in [3].

Note also that using grids with larger values of ρ would lead to jumps when switching levels. Rendering quality and performances have a smaller granularity when ρ increases.

However, we recognize that a larger branching factor results in jump in memory consumption when switching levels, which may be problematic in certain context.

The precomputation of the displacement vectors consists of three multiplications per level, and $3\rho^3$ additions per level. This cost is negligable compared to the operations required for all the samples (hundreds of thousands).

5.5 Pre-rendering indexed materials

For many types of models, the number of materials is small, compared to the number of samples. In [2], shading is precomputed for each possible normal quantized direction; we use $2^{13} = 8k$ directions. For each normal index, we compute the shading per material, giving a table of evaluations of material properties for each lighting angle. Using this table, we can shade a sample simply with a look up table.

Given the color quantization on mobile devices, the normal representation we used proved to be largely sufficient. Indeed, the dynamic range available for colors is about 5-6 bits per component, the precision for luminance is thus about 0.01. If we want to represent highlights, with a Phong exponent of the order of 10, then we need a precision of about 0.001 for the dot product of the normals. We estimated (with random tests) the precision of quantization in Table 3, which justifies the choice of level 5 of normal quantization for 16 bits colors displays.

level	3	4	5	6	7
error	0.017	0.004	**0.001**	2.6 e-4	6.4 e-5

Table 3: dot product error given normal quantification.

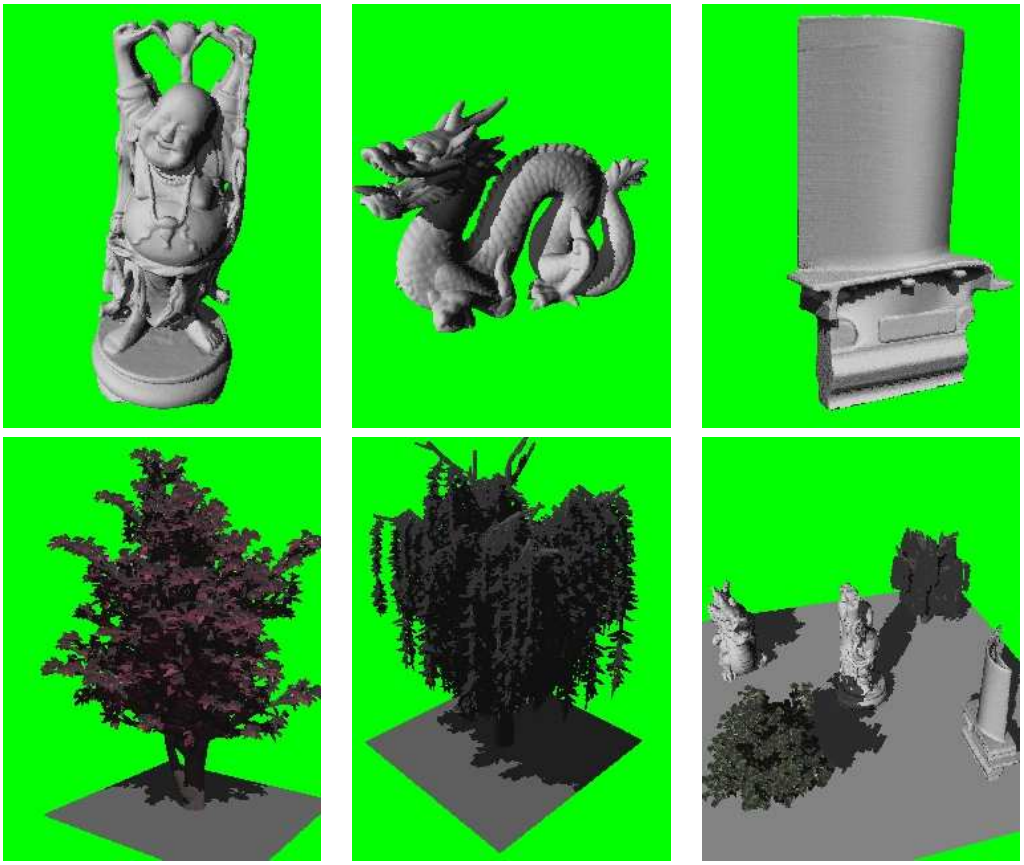


Figure 4: The five models we used (see Table 4 for statistics of the scenes). We have also constructed a "Big Scene", containing all five objects and a ground plane, represented as a point set.

6 Implementation and Results

In this section we present the results of our data structure construction and point-based rendering. The tri-grids are computed on a workstation, and transferred to the iPAQ using a PCMCIA Compact Flash card. The iPAQ we use has a 200Mhz processor with 64 Megabytes of main memory. In practice however we have about 8-9 Mbytes available for our data structure.

We have used five models, three from the Stanford database (Buddha, Dragon and Blade) and two tree models. We have also used the Stanford Bunny as a reference for our

explanation of the α parameter. All statistics are for 16 bit attribute samples, 13 bits for normals and 3 bits for a material index.

Please also see the accompanying video (with sound) which contains additional results, and examples of the system in action.

6.1 Implementation issues

Most mobile platforms do not have floating point units. As a result direct porting of graphics code will have significantly reduced performance. In addition, certain complex instructions may not be implemented; for example a division can be more expensive than a look-up in a precomputed table. This was confirmed on the platform we use.

We implemented fixed point arithmetic for the purpose of our renderer, i.e., multiplications and additions. We also implemented an approximation of the inverse up to a given precision using a look-up table. Since we use 32-bit fixed point numbers, a global look-up table could not be used. We use a shift on numbers instead, with table size depending on expected precision.

6.2 Basic Statistics

In Table 4 we present a set of statistics on memory usage of our data structures. We first show the number of polygons contained in the original model, and then show the number of samples created by the structure for different maximum levels of subdivision of the trigrid. For example, the total number of polygons contained in the Blade model is 1.76 million; at level 3 we have 17,000 samples, at level 4 we have 180,000 and at level 5, 1.7 million. Level 5 subdivision may not seem very useful, since it can even result in a higher number of points than the number of polygons. However, it can be seen as a very simple level-of-detail data structure, which allows flexible and efficient rendering. The cubic nature of the cells of our hierarchy affects the number of samples. We thus see that trees, which are more "cubic" than the blade for example, have a larger number of samples.

We next show the values of α (see Section 4) calculated on our tri-grids. The statistics for the Bunny illustrate that for closed and "well-behaved" models, α is equal to 9. For other models, for level 3, sampling is not fine enough, resulting in higher values. For level 5 we see that the values stabilize around 9. The tree models do not seem to "converge" to 9, due to the effect of the small leaves which create an "edge" effect in the data structure.

We next show the size of the tri-grid data structure when stored to disk. Note that the core data structure size increases by about 25% since we store additional pointers. This additional memory cost could definitely be reduced. As mentioned above, in practice we have about 8Mb available on the iPAQ, thus we can fit all models at level 4, or a combination of level 3, 4 and 5.

Model	polygons	Samples 3	Samples 4	Samples 5	α 3	α 4	α 5
Bunny	69 k	26 k	230 k	2.1 M	9.36	9.03	9.00
Dragon	870 k	18 k	170 k	1.5 M	9.75	9.19	9.04
Buddha	1.08 M	14 k	130 k	1.2 M	10.3	9.45	9.07
Blade	1.76 M	17 k	180 k	1.7 M	13.19	10.77	9.26
Arbre	540 k	40 k	370 k	3.2 M	11.45	9.18	8.63
Saule	420 k	45 k	430 k	3.4 M	13.62	9.62	7.84
Big Scene	4.67 M	134 k	1.28 M	11.0 M	-	-	-

Model	wrl.gz file	File Size 3	File Size 4	File Size 5
Bunny	858 kb	76.8 kb	698 kb	6.28 Mb
Dragon	8.90 Mb	54.5 kb	506 kb	4.60 Mb
Buddha	11.0 Mb	41.5 kb	396 kb	3.65 Mb
Blade	14.4 Mb	46.2 kb	520 kb	5.06 Mb
Arbre	8.53 Mb	114 kb	1.1 Mb	9.77 Mb
Saule	9.31 Mb	121 kb	1.28Mb	10.7 Mb
Big Scene	52.14 Mb	377 kb	4.5 Mb	40 Mb

Table 4: Basic model statistics. We show the number of polygons and the number of samples generated for different maximum levels of refinement (3, 4 and 5) for the tri-grid. The α parameter is computed for each of these levels, as well as the file size for the tri-grid structure.

Model	Points	splats	shadows	One-Pass
Buddha (4)	5.49 (4.15)	3.47	2.65	2.99
Buddha (5)	0.91 (0.63)	3.33	2.46	2.85
Blade (4)	4.71 (3.30)	2.63	1.99	2.32
Blade (5)	0.67 (0.47)	2.40	1.89	2.21
Big Scene	0.62 (0.30)	2.38	1.83	2.11

Table 5: Rendering statistics for five variants of our rendering algorithm (in frames per second). Points corresponds to direct display of points (shadowed version in parentheses), splats is the rendering multi-level algorithm with splatting, shadows is the same with two-pass shadows and 1-pass shadows uses the one-pass shadow algorithm.

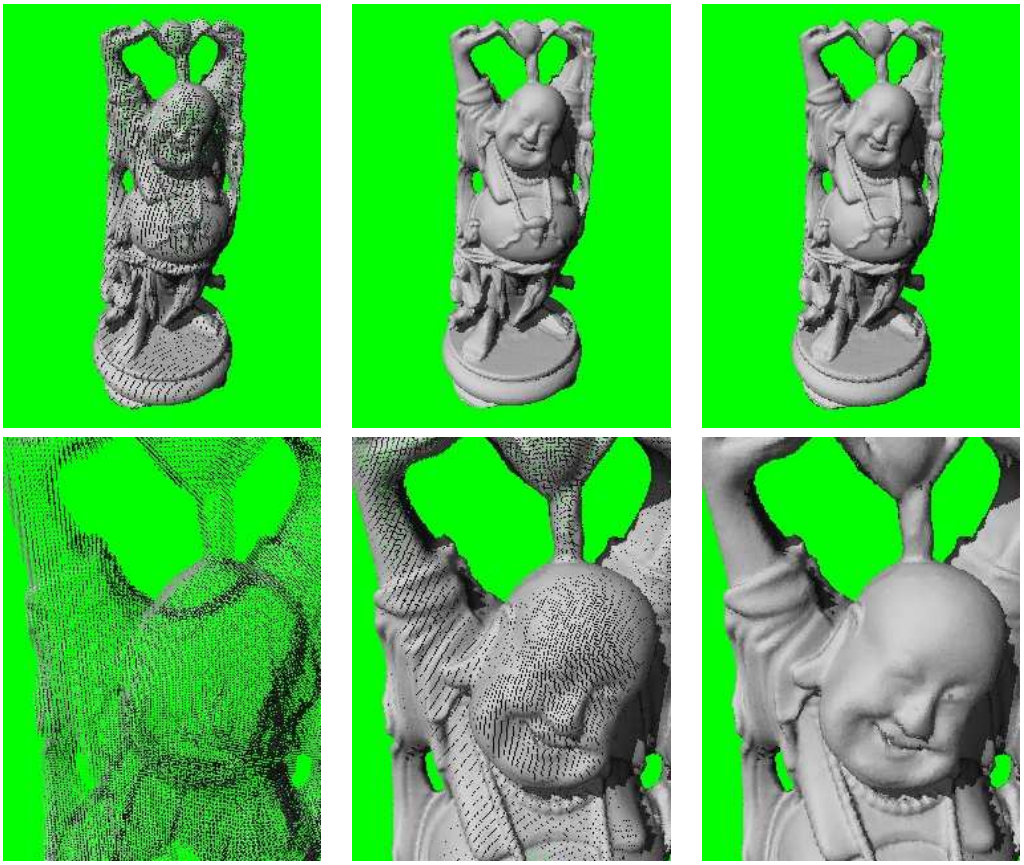
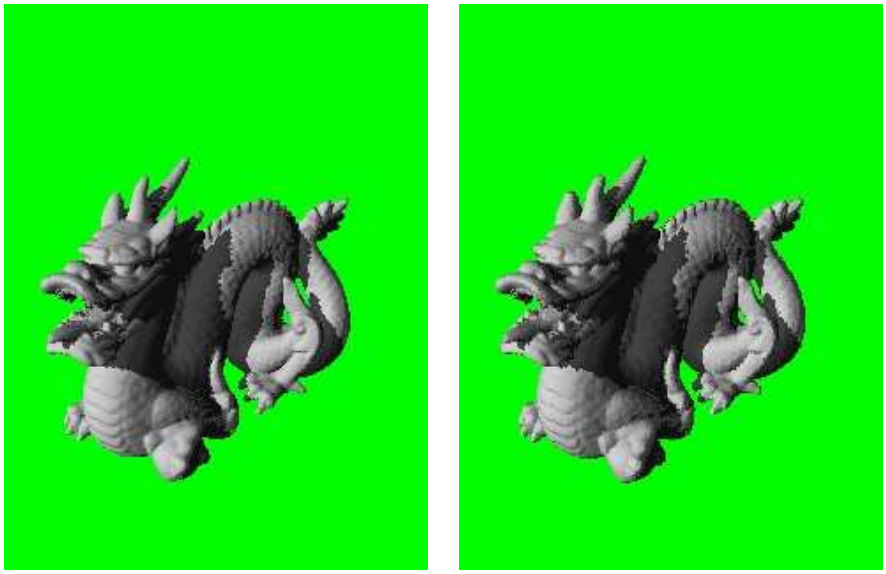


Figure 5: Above, a far view of the Buddha model shown (left) with points at level 4 (4.15 fps) (middle) with points at level 5 and (0.63 fps) (right) with splats at level 5 (2.85 fps). Below, a close view of the Buddha model shown (left) with points at level 4 (4.15 fps) (middle) with points at level 5 and (0.63 fps) (right) with splats at level 5 (1.42 fps). Undersampling problems are evident at level 4 subdivision when using points only. At level 5, the increase in frame rate is notable.

6.3 Rendering Performance and Shadows

In Table 5 we show statistics for the four variants of the rendering algorithm using the tri-grid structure. The statistics are average frames per second on the iPAQ. An example of the kind of view chosen is shown in the top row ("far view") of Fig. 5. As expected, overall rendering all the points is more expensive than the multi-level display, especially at maximum subdivision level 5. Nonetheless, for level 4 subdivision, the expense of computing the splats



0.55 fps 2.28 fps

Figure 6: Quality and speed of shadows. View of the Dragon model shown (left) with points at level 5 with shadows and (right) multi-level display with one-pass shadows multi-level. Notice that shadows generated with multi-level rendering are practically indistinguishable from shadows generated with full point resolution.

can outweigh the gain from multi-level rendering (Buddha(4) and Blade(4) in Table 5). Note however (Fig. 5 upper left) that Buddha at level 4 has visible undersampling artefacts with pure point rendering.

In Fig. 5, we show the problem of undersampling for a close view of a model, which is remedied using splats. In addition, we show the effect on frame rate of multi-level display. Recall that multi-level display is possible because we store intermediate samples.

7 Conclusion

In this paper we have presented a general framework for point-based sampling of complex models, which we call ρ-grids. We have performed a theoretical analysis of the storage requirements of these structures, and we have shown that the tri-grid is the most memory efficient when we store sample attributes for intermediate nodes.

We have used the tri-grid structure for efficient rendering of very complex models on memory, computation and display-size limited mobile devices. We have introduced a flexible multi-level rendering algorithm which allows efficient rendering when zooming into the

model, and an efficient one-pass shadow mapping algorithm. Our implementation on a 200 Mhz Compaq iPAQ PDA allows the display of 1.3M point-based representation of a 4.7 million polygon model at 2.1 frames per second.

In future work, we will be examining better reconstruction algorithms, and a progressive transmission approach which will take into account network parameters. We will also investigate hybrid rendering combining different representations such as polygons, lines and points, taking into account the specific constraints of mobile devices.

8 Acknowledgements

The first author is supported by an AMX doctoral fellowship. The authors wish to thank Alexandre Olivier-Magnon for modelling the tree; the Buddha, Bunny, Blade and Dragon model are from the the Stanford 3D Scanning Repository http://graphics.stanford.edu/data/3Dscanrep. Thanks to M. Stamminger and F. Durand for their insightful comments on an early draft.

References

[1] M. Alexa, J. Behr, D. Cohen-Or, S. Fleishman, D. Levin, and C. T. Silva. Point set surfaces. In *IEEE Visualization 2001*, pages 21–28, 2001.

[2] M. Botsch, A. Wiratanaya, and L. Kobbelt. Efficient high quality rendering of point sampled geometry. In *Rendering Techniques 2002 (Proceedings of the Eurographics Workshop on Rendering 02)*. Springer Verlag, 2002.

[3] F. Cazals and C. Puech. Bucket-line space partitionning data structures with applications to ray-tracing. In *Symposium on Computationnal Geometry*, 1997.

[4] B. Chen and M. Nguyen. Pop: A hybrid point and polygon rendering system for large data. In *IEEE Visualization 2001*. IEEE, 2001.

[5] J. Cohen, D. Aliaga, and W. Zhang. Hybrid simplification: Combining multi-resolution polygon and point rendering. In *IEEE Visualization 2001*. IEEE, 2001.

[6] O. Deussen, C. Colditz, M. Stamminger, and G. Drettakis. Interactive visualization of complex plant ecosystems. In *Proceedings of the IEEE Visualization Conference*. IEEE, October 2002.

[7] O. Devillers and P.-M. Gandoin. Geometric compression for interactive transmission. In *Proc. of IEEE Visualization 2000*, pages 319–326, 2000.

[8] A. S. Glassner. Space subdivision for fast ray tracing. *IEEE Computer Graphics and Applications*, 4(10):15–22, October 1984.

[9] J. P. Grossman and W. J. Dally. Point sample rendering. In *Rendering Techniques '98*, EG workshop on rendering, pages 181–192. Springer-Verlag, 1998.

[10] Intel Corp. *http://www.intel.com/design/pca/ applicationsprocessors/1110_brf.htm*, 2001.

[11] D. Jevans and B. Wyvill. Adaptive voxel subdivision for ray tracing. In *Proceedings of Graphics Interface '89*, pages 164–72, Toronto, Ontario, June 1989. Canadian Information Processing Society.

[12] M. Levoy and T. Whitted. The use of points as display primitives. Technical Report TR 85-022, Univ. of North Carolina at Chapel Hill, 1985.

[13] Fabrice Neyret. Modeling animating and rendering complex scenes using volumetric textures. *IEEE Transactions on Visualization and Computer Graphics*, 4(1):55–70, January–March 1998. ISSN 1077-2626.

[14] H. Pfister, M. Zwicker, J. van Baar, and M. Gross. Surfels: surface elements as rendering primitives. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 335–342. ACM Press/Addison-Wesley Publishing Co., 2000.

[15] S. Rusinkiewicz and M. Levoy. Qsplat: a multiresolution point rendering system for large meshes. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 343–352. ACM Press/Addison-Wesley Publishing Co., 2000.

[16] S. Rusinkiewicz and M. Levoy. Streaming qsplat: a viewer for networked visualization of large. In *Proceedings of the 2001 symposium on Interactive 3D graphics*, pages 63–68. ACM Press, 2001.

[17] M. Stamminger and G. Drettakis. Interactive sampling and rendering for complex and procedural geometry. In K. Myskowski and S. Gortler, editors, *Rendering Techniques 2001 (Proceedings of the Eurographics Workshop on Rendering 01)*, 12th Eurographics workshop on Rendering. Eurographics, Springer Verlag, 2001.

[18] M. Wand, M. Fischer, I. Peter, F. Meyer auf der Heide, and W. Straßer. The randomized z-buffer algorithm: Interactive rendering of highly complex scenes. In *SIGGRAPH 2001 Conference Proceedings*, pages 361–370, 2001.

[19] M. Zwicker, H. Pfister, J. van Baar, and M. Gross. Surface splatting. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 371–378. ACM Press, 2001.

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Unité de recherche INRIA Futurs : Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)
Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399