SEQUENTIAL POINT CLUSTERS: EFFICIENT POINT-BASED RENDERING METHOD FOR HUGE 3D MODELS

Sequential Point Clusters:
大規模モデルに対する効率的な
ポイントベーストレンダリングシステム

by

Yasuhide Okamoto
岡本　泰英

A Senior Thesis
卒業論文

Submitted to
the Department of Information Science
the Faculty of Science, the University of Tokyo
on February 9, 2004
in Partial Fulfillment of the Requirements
for the Degree of Bachelor of Science

Thesis Supervisor: Katsushi Ikeuchi　池内　克史
Professor of Institute of Industrial Science

ABSTRACT

Advancement in modeling technologies has enabled us to obtain 3D models composed of an extremely large number of polygons. Unfortunately, however, the conventional polygon-based rendering method is not rapid enough for rendering such models. As a result, we cannnot display these models in an interactive manner. In this thesis, we propose a system for effectively rendering such models based on using the Point-Based Rendering technique that considers multi-resolution representations.
Our system has the following characteristics: The data structure, referred to as Sequential Point Trees, realizes the multi-resolution representations. The structure enables our system to rapidly render the representations by using graphics hardware. Additionally, we propose a hybrid rendering method that combines point- and polygon-based rendering to improve the quality in rendering results. As a result, our system can rapidly render high quality models. In addition, we present the extension to Sequential Point Clusters, which is accomplished by using positional clustering, which can reduce the amount of data transfer.
We have implemented the system and verified the efficiency of our proposed algorithms by rendering models composed of polygons whose number is more than ten million. And we have searched the best level of clustering, and demonstrated the superiority of clustering over other methods.

論文要旨

モデリング技術の向上により、大規模な３次元モデルの生成が容易になってきた。しかし、こうして生成された大規模なモデルを描画する際、従来のポリゴンを用いたレンダリング手法では非効率的でありインタラクティブ性に欠けるという問題があった。そこで本論文では、多重解像度の考えに基づき、Point Based Rendering 手法を用いた効率的な描画システムを提案する。

本システムでは、多重解像度を Sequential Point Trees を用いて実現し、描画処理をグラフィックスハードウェアのみを用いて処理する手法に加え、描画プリミティブにポイントとポリゴンを混在させることで、高速かつ高精度な描画を可能としている。さらに我々はデータの幾何的な位置によるクラスタリングを行うことで、処理データ量を削減する Sequential Point Clusters への拡張を提案する。

実際に、面数が数千万である大規模なモデルを用いて本手法の実用性を検証するとともに、最適なクラスタリングレベルを探り、その有効性を明らかにする。

Acknowledgements

Thanks to Prof.Katsushi Ikeuchi for giving advices. Thanks also to Shuntaro Yamazaki for providing a great deal of knowledge and materials, and to Marie Elem and Jun Takamatsu for proofreading. In addition to them, everyone in the Ikeuchi laboratry for creating and providing many models, and giving us an excellent environment.

Contents

List of Figures

Chapter 1

Introduction

1.1 Background

In the computer vision field, there is an application that converts real objects into digital 3-Dimensional models. As one of our projects, we have been working on digitally preserving cultural resources. Our preservation objects includes the small figurines, the life-size statues, and the remains of large temples as particularly large models.

Recently, the improvement of camera scanning ability has enabled us to obtain very huge models with high fineness. This caused increasing of the size of models. For example, the model of a temple has over 10 or 100 million polygons. The data of such large models can be easily preserved by recent storage devices. But this increasing of data presents a serious problem. Such huge models cannot be rendered with traditional rendering methods, even if we uses modern workstations.

After obtaininging the data, we have a requirement to view the models on PCs. Normally, 3D data is stored as the set of polygons (triangles) which compose the model, and the 3D models are rendered by using the polygons. This polygon-based rendering is the major method of rendering, and can produce a model similar to the real object. The use of smooth shading enables the method to represent more complex shapes such as curved surfaces with a high degree of reality.

But polygon-based rendering is not available for very huge models because the polygon rendering requires complex computing to determine the inner pixels and the color. Due to the high cost of the calculations, the rendering performance rapidly declines as the model size increases. We need another method to render such huge models with higher efficiency, in particular for interactive work.

1.2 Related Work

One type of the solutions is the level-of-detail method. [LRC$^+$02] proposed the removal of the polygons whose projected size is too small to affect the image. [HOP96], [HOP97], and [KS96] presented the triangle mesh reduction which merges the smaller triangles to form larger ones. However their methods can render huge models because the cost of pre-computation and the CPU load is still impractical.

Another method is to use points, which are simpler primitives than polygons. The use of points as primitives was introduced in [LW85]. The concept of the point-based rendering is based on the following idea. In a complex scene, points which do not need complex computation are more suited to rendering because many of triangles become smaller than a single pixel. In [GD98] the modeling and rendering system using points as primitives was introduced.

Point-based rendering can be extended to level-of-detail. Fortunately, the cost of extension for points is less than that for polygons because the point-based method discards the connectivity information, and the data become simpler. QSplat[Lev00] and Surfel[PZvBG00] were introduced as a system based on point-based rendering and a hierarchy structure like level-of-detail. QSplat's system builds the hierarchy of bounding spheres whose leaves are vertices of the original model. In rendering, it traverses the hierarchy of resolution, selects proper resolution, and renders the bounding spheres. The system can skip rendering of invisible points, which are outside of the view frustum, or whose normal are away from the view vector. That is why it achieves interactive rendering. But point-based rendering has problems of image quality. When viewed very close to a model, the blocky shapes of points are projected as larger than a single pixel; therefore, the image quality decreases. To solve this problem, the POP system [Ngu01] was presented; this system uses both points and polygons as primitives, i.e.,hybrid rendering. The system uses points when the viewer is far from the model, but uses polygons when the viewer is close to it. The use of polygons decreases the efficiency of rendering a little, but the quality is always maintained.

Recently the performance of graphics processing units (GPUs) has improved, resulting in an increase in research of point-based rendering with GPUs. In [CH02] the points of a model are sorted back to front, and rendered by GPU without depth test, but with blending.

Sequential Point Trees[Sta03] is another method that uses point rendering and multiresolution hierarchy, but the algorithm and data structure are changed to a suitable form for GPU processing. The hierarchy is converted into a sequential list. In rendering, the system can render a model at the proper resolution by one sequential process for the list. Because the sequentiality is suitable for processing on GPUs, the system can render models at maximum efficiency.

The purpose of our work also includes interactive rendering for larger models than the size previous systems assumed. Our system is based on Sequential Point Trees, and we adopted hybrid rendering along with other methods. In addition to these methods, we give the efficient extension using clustering with respect to position of points – sequential point clusters. Although this clustering victimizes the CPU load, we save the amount of data to be transferred to the GPU. We proved that this positional clustering enables our system to render huge models with more efficiency than original Sequential Point Trees. In Figure 1.1, the examples of multiresolution rendering are shown. Our system achieved more efficiency and larger scale.

1.3 Thesis Overview

In this thesis, we describe our system in following section. In Chapter2, we propose the way to build a multiresolution hierarchy, sequentialize and render it by using the Sequential Point Trees algorithm, and hybrid rendering. In Chapter3, we propose the concept of Sequential Point Clusters, describe performance of our system and experimental results. In Chapter4, we describe how we verified the efficiency of our system and the best clustering level.

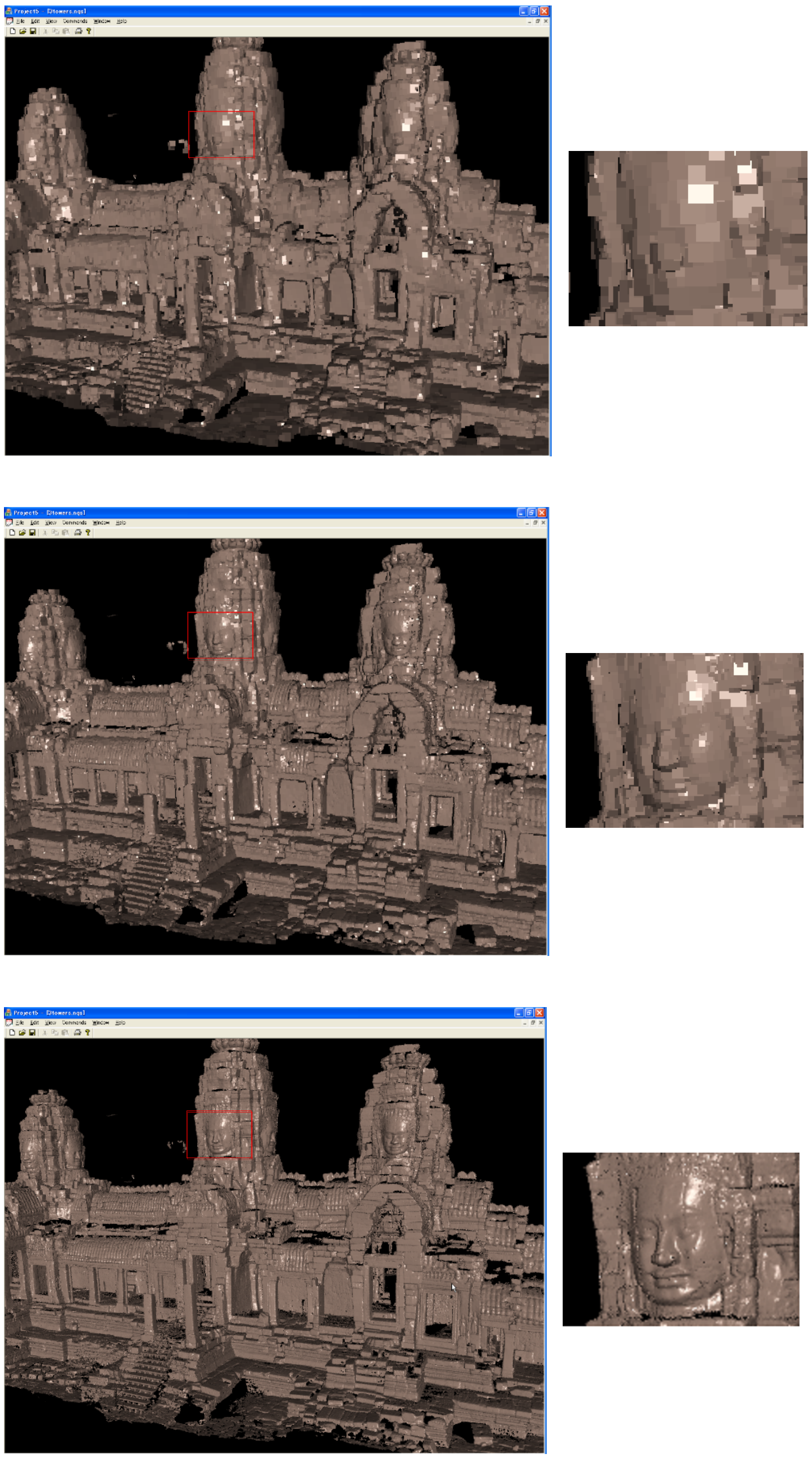


Figure 1.1: Multiresolution rendering : from top to bottom, the resolution increases. The images on the right are enlarged views.

Chapter 2

Data Structures and Algorithms for Rendering

Our system has three major characteristics. First, it has a multiresolution hierarchy structure. Second it can render models with a large of meshes at high speed, Third, it can render high quality images by hybrid rendering which uses polygons and points.

In this chapter, we describe concrete implementation of building a hierarchy structure, converting it to Sequential Point Trees, rendering, and hybrid rendering.

2.1 Data Structures of Multiresolution Hierarchy

Our system uses a multiresolution hierarchy structure, which is built from mesh data in advance of rendering. In building the hierarchy, we use QSplat[Lev00]'s algorithm. The algorithm creates a tree whose nodes are bounding spheres [RW80], [AK89] which have their center, normal, radius, and optional information as parameters.

When rendering, QSplat traverses this hierarchy from the root to leaves. If a current node is located outside of the view frustum or if its normal is away from the viewer, the node and its subtree can be skipped. And if a current node is far enough from the viewer that the projected size of the node on screen is below a constant value (usually 1 pixel), the node is drawn and its subtree can be also skipped. In the case that the node traversed is a leaf node, the system draws the node. But we do not use QSplat's algorithm of rendering; rather, we use the building hierarchy algorithm.

Building a tree algorithm begins with the creation of mesh data into bounding spheres, which are leaf nodes of the hierarchy. Our system converts one polygon into one sphere, the convertion algorithm is described in the latter section.

Our algorithm builds up the hierarchy by splitting the bounding box, including a

```
BoundingSphere BuildTree(BoundingSphere leaves[begin ... end])
{
  if (begin == end)
    return leaves[begin];

  else
  {
    middle = PartitionBoxAlongLongestAxis(leaves[begin ... end]);
    lefttree = BuildTree(leaves[begin ... middle]);
    righttree = BuildTree(leaves[middle+1 ... end]);
    return CombineBoundingSphere(lefttree, righttree);
  }
}
```

Figure 2.1: Building a tree algorithm

set of bounding spheres as the pseudo code shown in Figure 2.1.

The bounding box is split into two child boxes along the longest axis of the box, and two sub-bounding spheres are recursively computed from sets of bounding spheres included in the two boxes. The parent sphere is generated from the children spheres; the sphere's center, normal and other parameter are the average of those children. The radius is large enough to include children spheres for the purpose of averting holes when rendering. In the above algorithm, each node is generated by combining two children, but this increases the number of inner node more than necessry. That is why QSplat set the branching factor of the tree up as about 4. Our system also does that.

2.2 Algorithms using Sequential Point Trees

In rendering models, we use the Sequential Point Trees [Sta03] algorithm. The algorithm uses multiresolution hierarchy along with QSplat, but Sequential Point Trees sequentializes the hierarchy structure. That is why the algorithm can render at high efficiency by using graphic hardware at maximum efficiency. The sequentialization enables us to save memory on the building tree process. Because the sequentialization process is to decompose and rearrange nodes of the tree, the system does not have to store the entire data of the tree in the original order until the end of the building process. The advantage is very efficient, building a too large number of nodes of the data to store the entire data on system memory. The algorithm of sequentialization and rearrangement is written later.

But sequentialization has several problems. One is that the sequentialization of the tree does not enable us to quantize the tree in the QSplat's way. QSplat quantizes the tree nodes' location and radius as properties relative to the parent's. The properties are decoded on recursive processing in rendering. The Sequential Point Trees algorithm does not process trees recursively, but instead processes them sequentially, so that QSplat's quantization by using the relative value of the parent node is impossible. In this thesis, we do not discuss quantization of data.

The preprocess of rendering begins with building the multiresolution hierarchy as QSplat. In addition to this, we need to sequentialize the hierarchy tree. The algorithm is described in this section.

2.2.1 Definition of Errors

Each inner node in the tree is approximated to the set of the children nodes, so that such nodes have errors. The Sequential Point Tree gives the error value as optional information to all nodes of the tree. The value of a node is ditermined as the relative error to the set of children nodes. If the apparent error of the node on screen is over the threshold, the node is not rendered, and the children nodes are recursively tested.

The error is defined by two values: the perpendicular error e_p and the tangential error e_t. These values are defined in the next paragraph.

Perpendicular Error

Here, we treat the bounding sphere as the bounding disk whose plane is vertical to the normal. The perpendicular error is the minimum distance between two planes parallel to the disk that encloses all children disks. In other words, this error means spreading of the children disks set in the direction of the normal of the disk(see Figure 2.2). The error is computed by the following formula.

$$e_p = \max_i\{((\mathbf{p}_i - \mathbf{p}) \circ \mathbf{n}) + d_i\} - \min_i\{((\mathbf{p}_i - \mathbf{p}) \circ \mathbf{n}) - d_j\} \tag{2.1}$$

$$(d_i = r_i\sqrt{1 - (\mathbf{n}_i \circ \mathbf{n})^2})$$

The perpendicular error is projected onto the image in rendering. The projected error value $\tilde{e_p}$ is the value which depends on the normal of the disk $\mathbf{n}$ and the view textbftor $\mathbf{v}$ from eye to the disk. The value of $\tilde{e_p}$ is computed by the following formula.

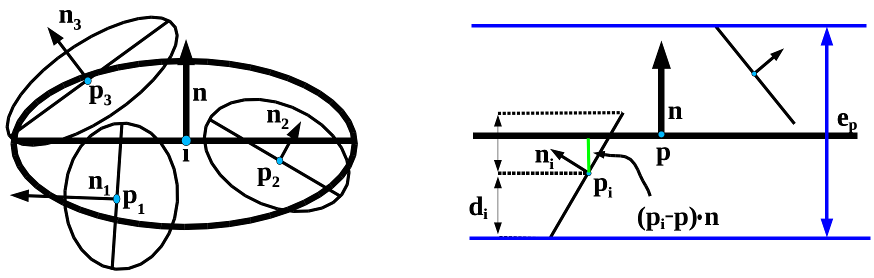


Figure 2.2: Perpendicular error means maximum spreading of the child disks in the direction of the normal.

$$\tilde{e_p} = \frac{e_p \sin\alpha}{r} \tag{2.2}$$
$$(r = |\mathbf{v}|, \alpha = \angle(\mathbf{v}, \mathbf{n}))$$

$\tilde{e_p}$ affects the silhouette of the rendering image.

Tangential Error

In contrast to the perpendicular error, the tangential error is the minimum distance between two planes vertical to the disk enclosing all children, so that this error means expansion of the children along the plane parallel as the disk. The value of this error is determined by computing the value that is the length of the disk's diameter minus the minimum width of children disks set ,as shown in Figure 2.3. In short, this error shows how large an area which is unnecessary the disk covers. If the computed value is under zero, this error is set to zero.

The tangential error is projected as well as the perpendicular error. The projected error $\tilde{e_t}$ is computed by the following formula.

$$\tilde{e_t} = \frac{e_t \cos\alpha}{r} \tag{2.3}$$

Geometric Error

We definite the geometric error e_g combined above two errors in the follow equation.

$$\begin{aligned} e_g &= \max_{\alpha}\{e_p \sin\alpha + e_t \cos\alpha\} \\ &= (e_p^2 + e_t^2)^{\frac{1}{2}} \end{aligned} \tag{2.4}$$

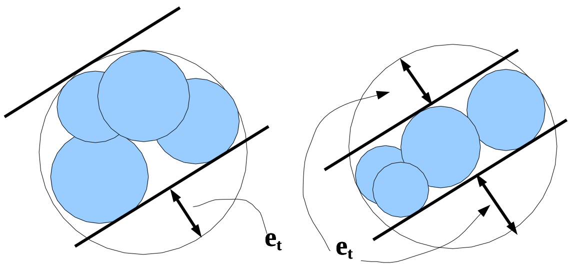


Figure 2.3: The tangential error means how much the parent disk is occupied by children.

$\tilde{e_g}$, which is the projected e_g in image space, depends on r as well as the above two errors, but is no longer dependent on the view angle α.

$$\tilde{e_g} = \frac{e_g}{r} \tag{2.5}$$

The upper bound of this error e_g is the diameter d of the disk. If e_g is computed to the value over d, we set e_g to d. e_g means how roughly the disk is computed from the children.

If we render a model representing the tree with a recursive algorithm like QSplat, we use the error defined above in the following way (see Figure 2.4). For every node of the tree, the projected error $\tilde{e}$ is recorded. In traverse, when the error of the current node is over threshold ϵ and the node is not a leaf, we traverse the children of the node recursively. Otherwise, we render the disk of the node.

Generally, the error is lower on the large and flat area than on the complex area. That is why the flat surface is rendered by large primitives and the complex part is rendered by dense primitives. In other words, the recursive test enables us to render models of high quality depending on the complexity of the area.

The threshold ϵ is a parameter which determines image quality and rendering speed. If the parameter is lower than 1, the size of the primitives are smaller than 1 pixel and image quality is enhanced. Otherwise, the quality is down, but the rendering speed increases. The user can determine the parameter, depending on the capability of the machine.

```
void TraverseHierarchy(TreeNode node)
{
  **if** (node is a leaf node **or** node.$\tilde{e_g}$ < $\epsilon$)
  {
    draw(node);
    **return**;
  }
  **else**
  {
    **for** (i = 0 ... the size of node.children)
      TraverseHierarchy(node.children[i]);
    **return**;
  }
}
```

Figure 2.4: Rendering algorithm

2.2.2 Sequentialization

The GPU sequentially processes the vertex data in the array. Therefore, the above recursive algorithm is unsuitable for processing on the GPU. Here, we need to convert the tree data to the array data, and the recursive procedure to a sequential one. In this section, we describe the sequential rendering algorithm runnable on the GPU at high speed and efficiency.

In the above recursive rendering, we use $\tilde{e_g}$ to test whether the node is rendered. However we can simplify this test. The check equation is $\tilde{e_g} = \frac{e_g}{r} < \epsilon$. ϵ is a constant value and e_g is determined for every node in building tree. So we can give every node another check value instead of e_g, which is a minimum distance r_{min} from a viewer to the node to pass the test. r_{min} is computed by the following equation,

$$r_{min} = \frac{e_g}{\epsilon} \tag{2.6}$$

By recording r_{min} on every node the recursive test is simplified to whether $r > r_{min}$ or not.

The r_{min} test enables us to discard too large nodes whose r are less than r_{min} of that even if in sequential processing. However, we have not discarded too detailed nodes yet. In sequential processing, we need further information for the purpose of discarding those nodes, whether the nodes have possibility to be refered in recursive processing. We can test this by checking r_{min} of the parent node. If r_{min} of the parent node is more than r then, there is the possibility. So, we need to give every node the

r_{min} of the parent node in order to discard too detailed nodes. This parameter is defined as r_{max} of the node. If the node is the root of the tree, r_{max} is set to infinity. In consequence we have only to perform the following procedure for every node to render the image sequentially. We draw the node if $r \in [r_{min}, r_{max}]$. The GPU can do so simple tests during the rendering process for all points, so all the CPU has to do is transfer the sequential list to the GPU. Therefore, we cannot assign CPU but GPU to the most of rendering process, so that the CPU load is low.

2.2.3 Optimization

The above sequentialization enables us to simplify the rendering process and to use the GPU at maximum efficiency. However, the list can be longer which is constructed from a model with a large number of meshes, so the transfer and processing of the list decreases efficiency. In this and the next section, we describe the optimization that enables us to process the list at low cost even if it is very long.

The sequential list does not need to have information of every node's children and parent, so the list can be rearranged in an efficient order. By sorting the list for r_{max} in descending order as Figure 2.5, the process of the list is simplified. If we encounter the node whose r_{max} is less than r in checking from beginning of the sorted list, we do not have to check the remaining nodes because r_{max} of those nodes are also less than r and unable to pass the r_{max} test (see Figure 2.5). That is why we can skip the rest of the nodes.

But r is not constant for every node of a model, so r must be conservetively estimated, in short, at the minimum of r, $\min\{r\}$, in above r_{max} preprocessing.

In our implementation, we search the first node whose r_{max} is less than $\min\{r\}$ by binary search before transfering list. After that, we transfer the nodes of the sorted list to the GPU, just from first to last node whose r_{max} is more than r. In addition to this, every node is checked by r_{min} and r_{max} test on the GPU.

Instead of the above simple r_{max} test, we can use the simple r_{min} test. But we get less benefit from the simple r_{min} test than that of r_{max}. The lower we descend the hierarchy, the larger the number of the nodes becomes. That is because r_{max} test can cull more nodes, which discards lower part of the hierarchy, than r_{min} test which discards the upper part.

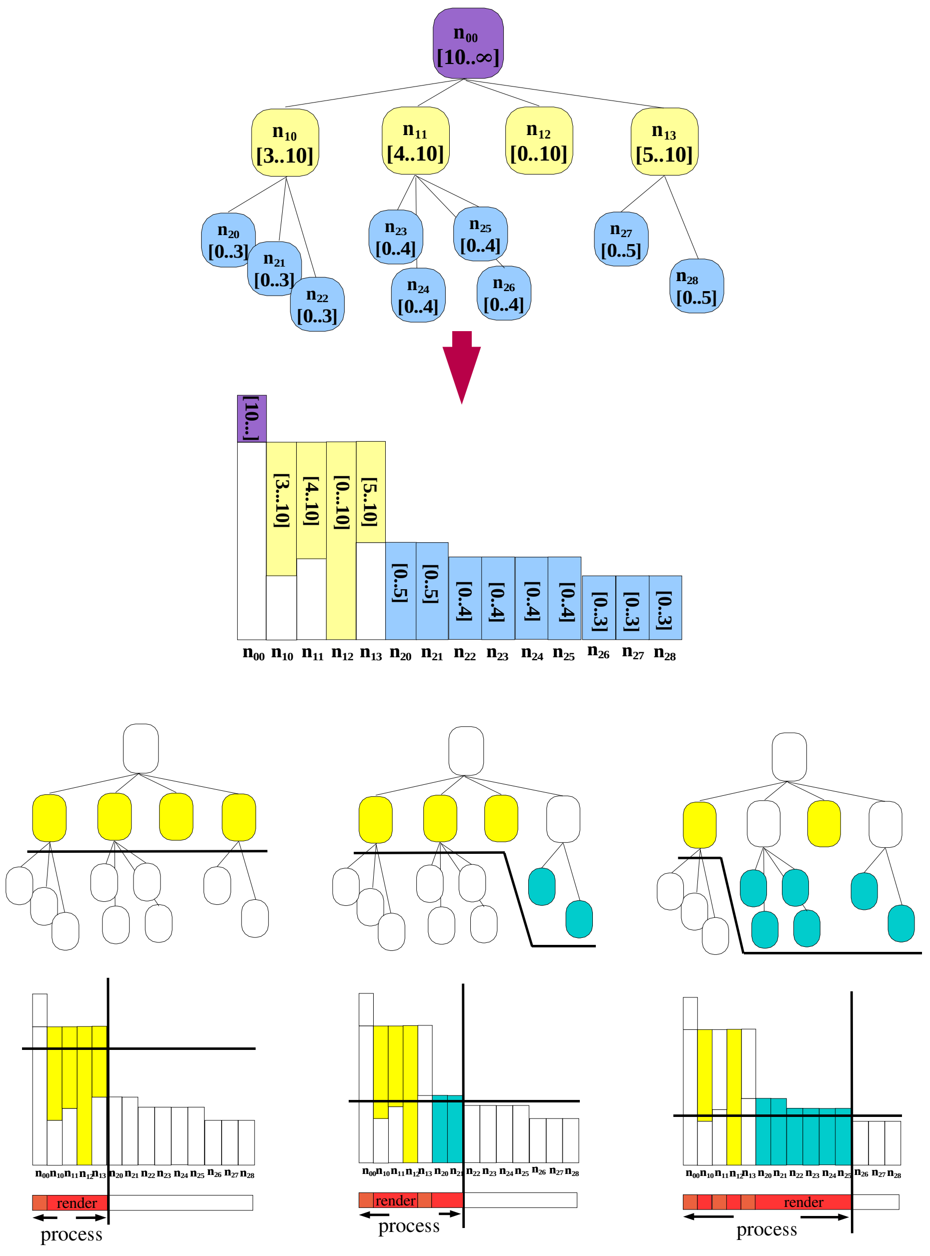


Figure 2.5: Sequentialization and optimized processing. Top : converting point tree to Sequential Point Trees. The nodes are with $[r_{min}, r_{max}]$. Bottom row : the visualization of the sequential processing for three different view distance. Tree is cut as above. And the bottom bars mean the range to be processed.

2.3 Extension

Although the Sequential Point Trees algorithm is very efficient, the peformance and quality can more increase by using several methods. In this section, we introduce the extended methods.

2.3.1 Normal Clustering

On QSplat's recursive procedure, we can cull the node and all nodes in the subtree if the current node is not visible because of out of view frustum or the normal away from the view vector. But Sequential Point Trees discard information of parent-child relation of the tree, so we cannot efficiently cull nodes in the way that QSplat does.

So, we split the node list into several clusters depending on the direction of the normal. We quantize all vectors to 128 and assign a cluster to a quantized vector. We distribute all nodes into the clusters whose vector are the nearest to the normal. We can cull nodes on the backface by discarding the clusters whose vector are away from the view vector before the rendering process.

On the other hand, clustering runs up the CPU load because it increases the number of preprocessing and transferring of the lists. But we can reduce the nodes which are transferred to GPU by almost 50%, whereas we get enough benefit to render at a higher speed by clustering.

2.3.2 Hybrid Rendering

Although point based rendering is effective in rendering models with a large number of small polygons, it has the problem of quality of the closeup image. Multiresolution hierarchy realizes rendering on dynamic resolution, but the density of the point does not increase more than that of the original model. If we view the model very closely, even the highest resolution points are rendered as blocky primitives. In addition to this, the set of points is rendered without interpolation between next points because it does not have connectivity information. For this reason, we adopt Hybrid Rendering based on POP[Ngu01] system, which uses both points and polygons as rendering primitives. Hybrid rendering is not only effective on the quality of close images, but it is also efficient on rendering models which have flat and large surfaces.

In the following section, we describe extension of preprocessing and rendering algorithms to hybrid rendering.

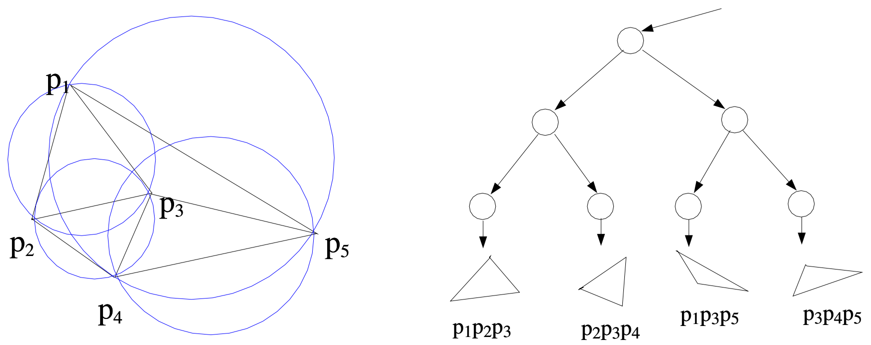


Figure 2.6: left : Generate bounding sphere from polygon. right : Building hierarchy for hybrid rendering

Preprocessing

We first extend the building hierarchy algorithm. While QSplat converts one vertex to one bounding sphere in the way of building a hierarchy, we must allocate one polygon to one sphere and give the mesh information as to the leaf of the tree.

The conversion from polygons to bounding spheres is done in the following way (See Figure 2.6).

If the triangle is an acute triangle, it is converted to the bounding sphere, whose center is the center of the circle that passes through all vertices of the triangle. Otherwise, if one of the the triangle's angles is larger than 90°, the center is that of circle which has the longest edge of the triangle as the diameter. The radius is set to that of the corresponding circle, and the normal is equal to the triangle's. By this calculation, we get the highest resolution bounding spheres. Additionally, we have to add polygon data to the tree. This is stored as a list of indices composing the polygon on the vertices table.

Rendering

We can extend the rendering algorithm by adding only the optional information r_{max} to the leaves of the tree which are polygons. If the projection size of the polygon is larger than the pixel size, we had better to render it by polygons. So, we define s as the longest edge of the polygon. If the projected length of s is longer than threshold ϵ, which means $\frac{s}{r} \geq \epsilon$, we render the polygon. We can compute r_{max} of the leaves from the formula as well as by the advanced way,

$$r_{max} = \frac{s}{\epsilon}. \tag{2.7}$$

And we rearrange the list of polygons in decreasing r_{max}, transfer to the GPU just from beginning to the last element whose r_{max} is more than $\min r$. We show a hybrid rendering example in Figure 2.7.

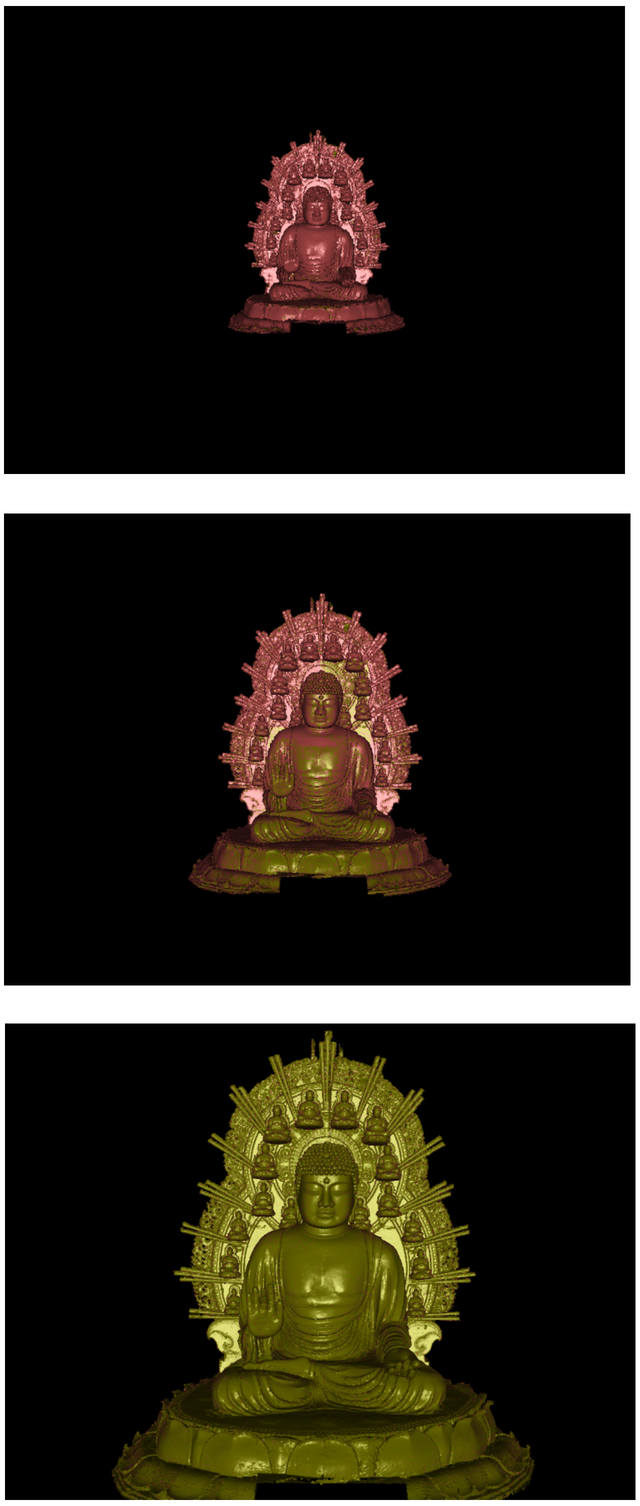


Figure 2.7: Hybrid rendering(red for points; yellow for polygons)

Chapter 3

Sequential Point Clusters

Our system is based on the above methods. Additionally, we present an extention to these methods. In the advanced section, we describe normal clustering to cover the incapability of backface culling in bulk on the Sequential Point Trees algorithm. But we lose much more advantage in addition to backface culling by sequentialization of tree structure. For example, we cannot do frustum culling in bulk as well as backface culling. Such a disadvantage is attributable to discarding the correlation between parents and children, ancestors and descendants of the tree. It solves these disadvantages to split Sequential Point Trees into clusters between whose elements there is correlation, like normal clustering. In this section, we present Sequential Point Clusters, which use clustering depending on the position of the element.

3.1 Positional Clustering

Positional clustering is effective in optimizing several processes. One of these gives the possibility of frustum culling. We split the tree into clusters in the bounding boxes, and we can skip rendering the cluster in which the bounding box is out of the view frustum. We get high efficiency when the region of the model is very small in the view frustum.

This benefit of the clustering is not only frustum culling. By clustering on position we can optimize the r_{max} test before transfering the trees to the GPU. This simple r_{max} test uses the minimum r which can be, but this is a roughly estimated result. Because such an estimation is the most conservative, the lists have more extra nodes to be transferred. If this problem is solved, we can reduce the length of the lists to transfer and do vertex processing for nodes of the tree. Clustering on position enables us to estimate r at higher precision for all nodes because the smaller cluster, the less

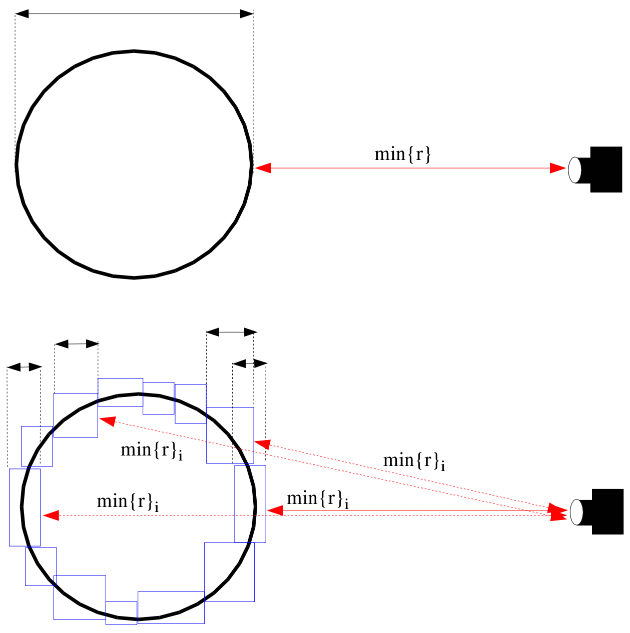


Figure 3.1: Positional clustering : This gives efficiency to simple r_{max} test. The differences decrease by clustering, between $\min\{r\}$ and nodes included a cluster.

the spreading of r is (see Figure 3.1). Note that this clustering is more effective for models with large mesh although it is less effective for small models.

In the simplified model, we explain this efficiency. We use a model whose tree nodes are located in homogeneous distribution, and the r_{max} of each node is also distributed homogeneously as shown in Figure 3.2.

First, we treat the data of the model as one sequential list as shown in the top of Figure 3.2. We apply a simple r_{max} test to the list. As the result of the test, the length of the list to process and transfer to the GPU is determined. We define the ratio of the length to the entire length of the list as p.

Second, we split the data into two clusters as shown in the bottom of Figure 3.2. We define the processing ratio of the front cluster as p_1 and that of the rear cluster as p_2. Because $\min\{r\}_1$ is equal to $\min\{r\}$, p_1 is equal to p. And because $\min\{r\}_2$ is greater than $\min\{r\}$, the number of nodes which pass the r_{max} test decreases and p_2 is less than p. Therefore, the p' which the processing ratio in this case is shown in the

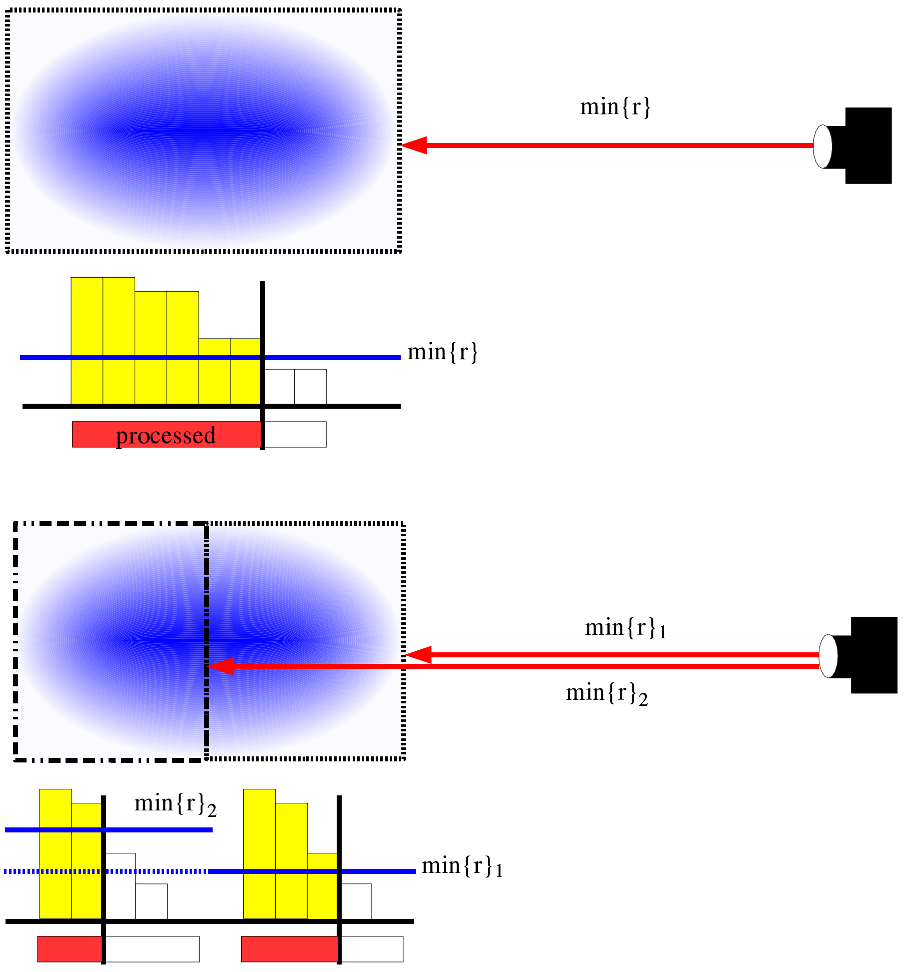


Figure 3.2: Positional clustering in a simplified condition. Top : treating the model as one list. Bottom : treating the model as two clusters. Because the $\min\{r\}_2$ is greater than $\min\{r\}$, the more nodes on the left cluster can be skipped.

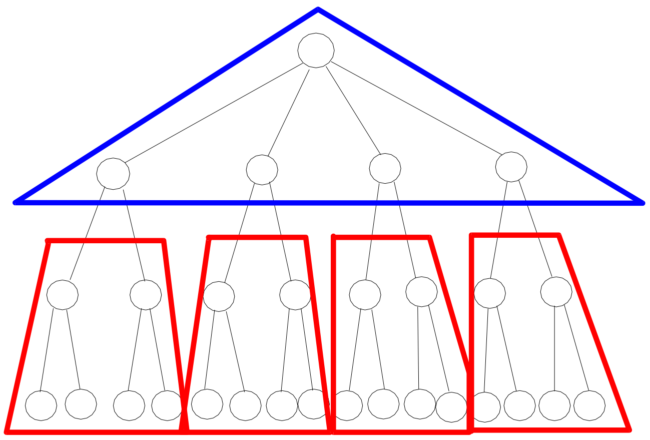


Figure 3.3: Hierarchical clustering : The numbers of the nodes included in each cluster are similarly even. This clustering presents efficient loading system.

following formula.

$$p' = \frac{1}{2}p_1 + \frac{1}{2}p_2 = \frac{1}{2}(p_1 + p_2) \leq \frac{1}{2}(p + p) = p \tag{3.1}$$

This means that the processing ratio by using clustering is less than that without clustering, clustering decreases the cost of processing and transferring.

We can recursively show the relation. Therefore, we can reduce more amount of data by more splitting.

3.2 Optional Extension

We add the extension to the clustering, which is splitting the tree hierarchically as shown in the Figure 3.3. This is also effective for large models. This clustering method generates clusters on several levels of resolution. When the viewer is very close to the model, we no longer have to render the low resolutional points. The hierarchical clustering enables us to skip to render the cluster whose nodes are lower resolution than the proper resolution. If we record $\min\{r_{min}\}$ of each cluster, we can determine the cluster by checking does $\min\{r_{min}\}$ of that whether it is the too low resolutional cluster to be rendered. We do not need to render the cluster whose $\min\{r_{min}\}$ falls under the following formula, $\min\{r_{min}\} > \max\{r\}$.

The such clustering gives us the above advantage, but this also raises problems as trade off of that, which are described in the section of normal clustering. The more the number of split clusters, the less the advantage of sequential process on the GPU is, and the more the load of the CPU is on the number of transferring and simple r_{max} test. But it is true that appropriate clustering gives efficiency especially for large models. So we experiment the rendering performance on several levels of clustering, and search for appropriate clustering levels in following chapter.

Chapter 4

Experimental Results

We implemented the system using methods described in Chapter 3; additionally we have optimized our implementation by using Sequential Point Clusters for the purpose of efficiency in processing, rendering, and saving memory. We verified the performance of our system, rendering on an ATI Radeon 9800 XT with DirectX, processing on a 3.2 GHz Intel Pentium4. We used vertex shaders to do sequential processing, but we did not use vertex buffers because of the impossibility of storing the data of large models on the memory of a graphics card. When rendering, we transfer the data every time.

We designed the system on the basis of interactivity. By using Sequential Point Clusters, we can easily change the resolution. This advantage gives us more interactivity. For example, when we are moving the view point, we want to render the model quickly. In contrast, we do not have to render it quickly when we look at the model at a fixed point. Therefore, when our system is implemented, the resolution is dynamically changed, depending on such situations.

In the following sections, we describe how we verified the performance, as the possibility of rendering large models, and the dependency of the performance on the level of clustering.

4.1 Performance of our system

We verified the performance of preprocessing and rendering. In addition to this, we discussed the rendering quality of our system. Figure 4.1 shows the results of performance for each model. SPC means the Sequential Point Clusters. In the () of SPC performance, we describe the percentage of the processed and transferred points to all points of the model, and the percentage of such polygons to all polygons of the model. The rendering performance is affected by not only size of model, but also by the parcentage.

4.1.1 Preprocessing performance

Before rendering, we need to preprocess, i.e. building Sequential Point Clusters. This time is not as important as rendering time, but it must be practical even if the original data is very huge. In our system, the preprocessing has two steps: the first is to read the original data (the file format is PLY), and the other is to build and store the Sequential Point Clusters. But we evaluate only the latter time.

On the above machine, preprocessing time for a model which has one million polygons is 43 seconds. The results for other models is also shown in Figure 4.1. These results are slower than QSplat [Lev00], QSplat reports 42 seconds as the processing time for two million vertices. However, that is natural because our system is based on sequential processing and hybrid rendering presents higher qulity and efficiency than QSplat. Considering that, this difference is not too large to be practical. In the following section, we describe the relationship between preprocessing time and size of models.

4.1.2 Rendering performance

Next, we verified performance of rendering for the same models. These performances of rendering are also depicted in Figure 4.1. Our system can send 20 million nodes to the GPU per second. Its result is inferior to the performance of Sequential Point Trees[Sta03]; it can send 77 million nodes per second to the GPU. But the system of Sequential Point Trees stores all geometry data in the graphics card memory. It does not need to transfer data. On the other hand, our system assumes treating too huge an amount of data to store in graphics memory. That is why our system has a few disadvantages, but it can render larger models than the system using Sequential Point Trees. The performance of our system is efficient enough to keep interactivity. Additionally, if we make vertex programming more sophisticated, the performance can be improved.

4.1.3 Rendering quality

Our system renders models on the basis of the spirit of hybrid rendering. We compare the rendering quality between images rendered by several methods; our system, only points, and only polygons. In Figure 4.2 we showed the images. We observed that the image rendered by only points is noisy and of low quality because we can

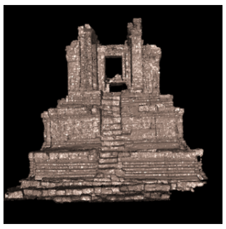
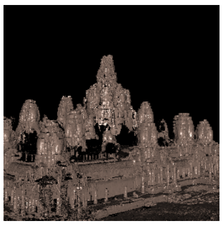
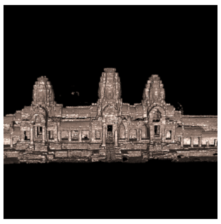
model	happy buddha	bayon face	kyo	towers	3towers
Input polygons	1,087,716	5,922,790	9,162,909	13,939,070	18,132,893
Preprocessing time	42.8s	125.3s	201.0s	377.2s	471.5s
Rendering by SPC	19.7fps(53.3/23.4%)	8.5fps(18.9/4.1%)	14.5fps(10.3/0.0%)	15.0fps(6.4/0.0%)	10.0fps(5.1/0.0%)
by Point	15.5fps	3.4fps	2.3fps	-	-
by Polygon	5.2fps	0.8fps	0.39fps	-	-
					

Figure 4.1: The preprocessing and rendering performance of five models. The mean of this table is described in Section 4.1.

find many splats with a blocky shape over the entire image. On the other hand, we observed that such blocky splats are much less than on images rendered by our system. This is because our system adopts hybrid rendering. We can conclude that our system is of higher quality than point-only rendering. But blocky splats remains on the area near the silhouette yet. Although our system is inferior to polygon-based rendering on this point, we cannot easily find the difference between images in their real size. In addition, we improve the quality by using Gaussian circles as primitives.

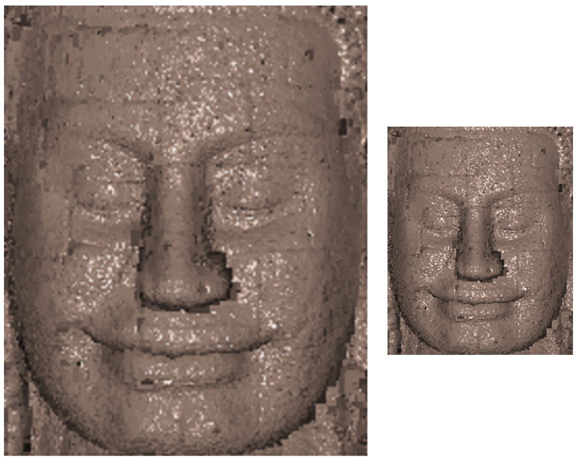
4.2 Performance for huge models

One characteristic of our system is to be available for larger models even if the size is much larger than other system. Preprocessing time is practical and rendering is real time. In this section, we demonstrate how efficient our system is for huge models.

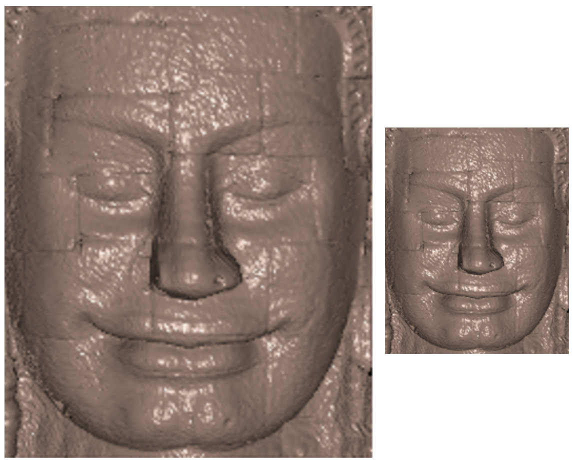
First, we verified the time of preprocessing for models of different sizes, from 1 million to 10 million polygons. Figure 4.3 shows the preprocessing time. The relation of increase of time and that of the number of polygons can be approximated linearly. That is why the time is under several hours even if the number of polygons is over hundreds of million. Additionally, memory needed for preprocessing is not too large. These things mean that our system can do practical preprocessing for hundreds of millions of polygons.

Second, we verified the performance of rendering for models of different sizes. The rendering performance is shown in Figure 4.4. From this result, we see a decline of performance of point and polygon from beginning to end. In particular, with regard to polygons, the frame rate is under 1.0 since the polygon number is over 5 million, which means that there is little interactivity. On the other hand, the results of our system are different from those methods. Although the performance decreases until the size becomes 1 million, that of interactive rendering starts increasing, and that of static rendering becomes roughly flat or slightly increases after 4 million. The reason for increasing performance is that the rendering resolution is less and the number of primitives, in particular polygons, decreases as size increases.

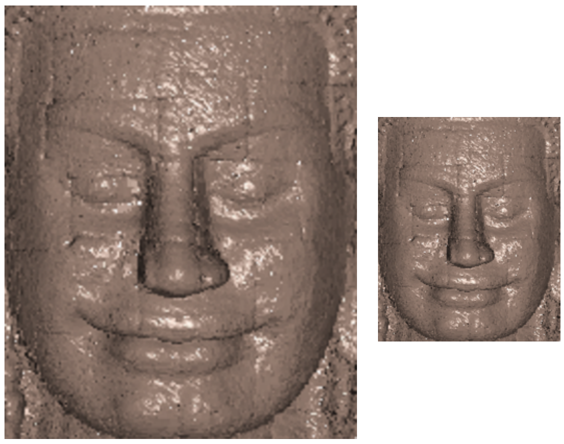
The result is that our system can render very large models at particular performance. No matter how large a model is requested, our system can render models by decreasing resolution.



(a)This is the image rendered by Point-based rendering.



(b)This is the image rendered by Polygon-based rendering.



(c)This is the image rendered by Sequential Point Clusters system.

Figure 4.2: Rendering quality by three methods. Left : the image in twice size of original. Right : the image in original size.

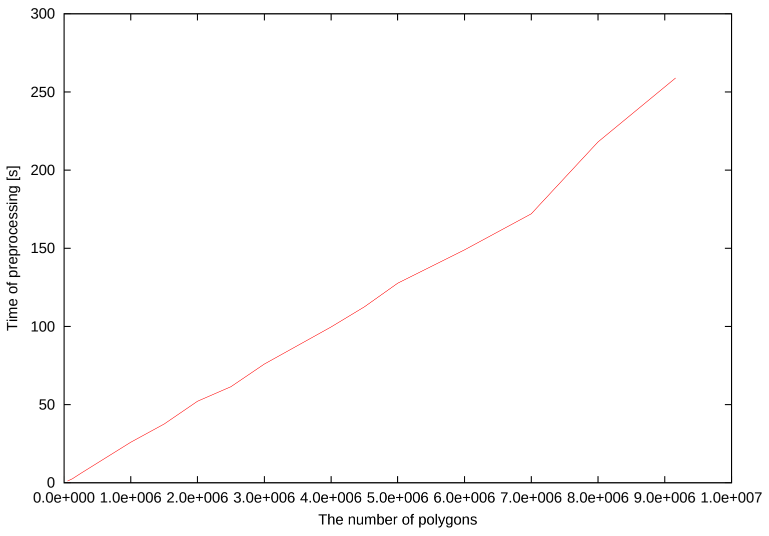


Figure 4.3: The preprocessing time to the size of the model.

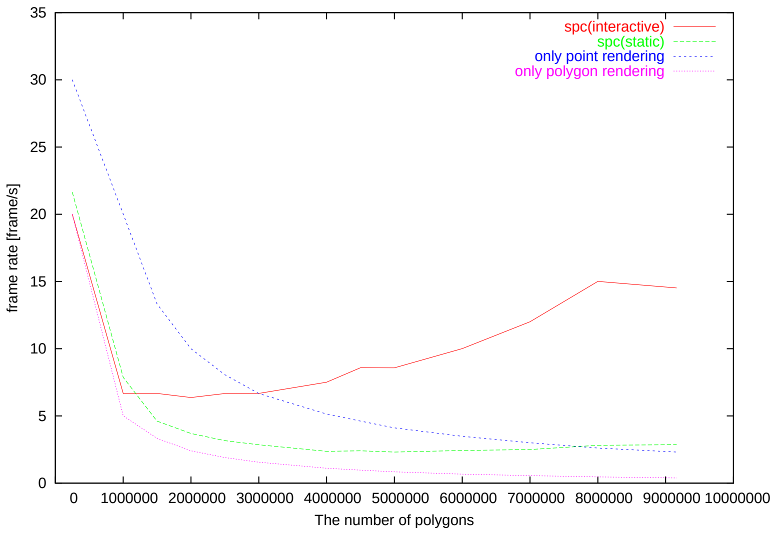


Figure 4.4: The performance to the size of the model by several ways. Rendering by our system , or point- and polygon-based rendering. “Interactive” means rendering at low resolution on our system, and “Static” means rendering at high resolution on our system.

4.3 The Evaluation of the Clustering Level

The majority of the rendering time is spent on vertex processing, checking the r_{min}, r_{max} test and computing the projected position, size, and color for every transferred node on GPU. This process cannot be tuned easily; therefore, we need to reduce the amount of transferred data to get better performance of rendering. The purpose of clustering is for that, but we reduce the advantage of sequential processing on the GPU by splitting the sequential list into more clusters. In this section, we compared the rendering performance of the data built by using different levels of clustering, and search for the best level of clustering.

In this experiment, we use three models which we split into clusters. The number of nodes included in each cluster is approximately from 10 thousand to 2 million. We measured the frame rate when rendering each model without hybrid rendering. The changing of the rendering performance is shown in Figure 4.5.

We found that the peak of performance exists when the number of nodes included in one cluster is between 100 thousand and 1 million. In addition, we saw that the range of peak decreases as the size of the model increases. The largest model of the three has a peak of 500 thousand.

When the number of nodes in 1 cluster is 500 thousand, the number of processed and transferred nodes at a time is in the thousands because the nodes in each cluster are still split into 128 clusters based on the normal. The fact, that peak range is from 100 thousand to 1 million, shows that the best number of processing and transferred primitives is about from 1 thousand to 10 thousand.

The reason why the range of peaks decreases is a bottleneck of the CPU load. The variation of the number of clusters become rapid as the size of the model increases. The rapid change of the clusters' number generates a sensitive change of the CPU load; checking simple r_{max} test and transferring data. Therefore, we need to pay more attention to the number of clusters for clustering a large model. On the basis of Figure 4.5, we know that the size of cluster had better been set to about 500 thousand as the model size is larger.

As a conclusion of this experiment, we find that the peak of performance is a changing level of clustering. It proves that the best clustering level exists, which is balanced in the processing and transfering traffic in a time, and in CPU load. We should set the cluster's size on the range of the peak. But because the range is shortened as the model size grows, it becomes hard for the level to be tuned. The best level can be on which the cluster size is 500 thousand from our results. And the performance of the

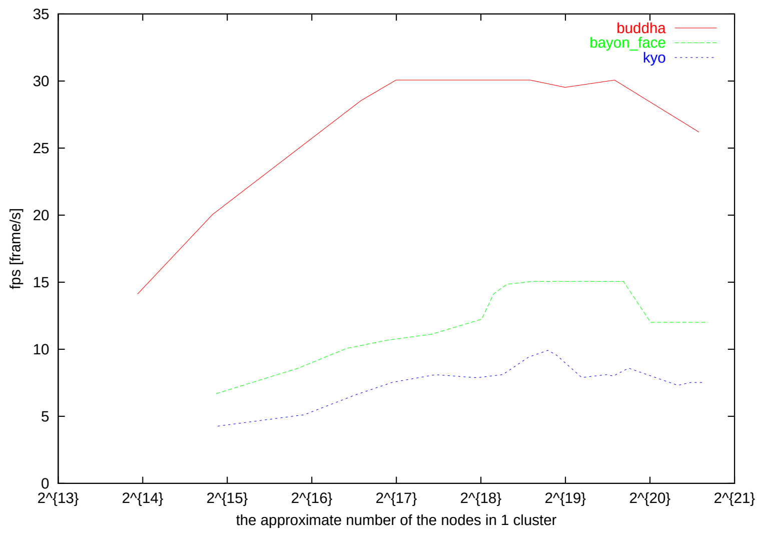


Figure 4.5: The performance to the level of clustering. The used models are Happy Buddha, Bayon face, and Kyo. Their size is shown in Figure 4.1.

peak is better than Sequential Point Trees without positional clustering.

Chapter 5

Conclusion

5.1 Summary

In this thesis we introduced the interactive rendering system using Sequential Point Clusters. This system is based on point-based level of detail with multiresolution hierarchy. The method enables us to render large models at enough resolution by the minimum rendering process. For the improvement of image quality, we adopted hybrid rendering which appropriately uses points and polygons as primitives depending on the position of a viewer. At the same time, we use the extended version of Sequential Point Trees. Sequential Point Trees can sequentially process the hierarchy data of models on the GPU by converting the hierarchy data to sequential lists. This algorithm can reduce the amount of the data to be processed by rearrangement and simple test. We added positional clustering to this algorithm. By this extension, we can save the transferred and processed data.

5.2 Contribution of this Thesis

We discovered the improvement in the performance by using Sequential Point Clusters which we proposed in this thesis. This enables us to achieve more performance than simple Sequential Point Trees. The performance of the system is interactive enough to operate it. No matter how large the viewed model is, our system can render it without keeping viewers waiting.

5.3 Future Work

Our system still has several points to be improved. The major ones are the following:

- We need compression of data. The larger the size of the model is, the higher the necessity of compression is. Our system cannot quantize the data like QSplat because of the loss of tree structure. But we search the new quantization, for example, by using clustered information.

- We want to make the algorithm of building hierarchy more sophisticated. Our algorithm is based on QSplat, but the algorithm using bounding boxes remains to be improved. We want the algorithm to build a tree suitable for high quality rendering and high compression.

- The improvement of the splatting method can present higher quality images than the current system. Our current system uses squares as rendering primitives. Although splatting using squares is very lightweight, the rendering quality still has problems, in particular with edges and silhouettes. Several researchers have proposed methods for higher quality point-based rendering. We want to adopt a method which is both efficient and high quality with the GPU.

Our system will be more efficient for the viewer by the addition of the above improvements.

References

[AK89] J. Arvo and D Kirk. A survey of ray tracing acceleration techniques. *An Introduction to Ray Tracing*, 1989.

[CH02] L. Coconu and H.-C. Hege. Hardware-accelerated point-based rendering of complex scenes. In *Rendering Techniques 2002 (Proc. Eurographics Workshop on Rendering),Springer*, pages 41–51, 2002.

[GD98] J. Grossman and W. Dally. Point sample rendering. In *Proceedings of Eurographics Rendering Workshop*, 1998.

[HOP96] H HOPPE. Progressive meshes. In *Proceedings of SIGGRAPH 96, Computer Graphics Proceedings*, pages 99–108, 1996.

[HOP97] H HOPPE. View-dependent refinement of progressive meshes. In *Proceedings of SIGGRAPH 97, Computer Graphics Proceedings*, pages 189–198, 1997.

[KS96] G. KLEIN, R. LIEBICH and W STRASSER. Mesh reduction with error control. In *IEEE Visualization '96*, pages 311–318, 1996.

[Lev00] Szymon Rusinkiewicz. Marc Levoy. Qsplat:a multiresolution point rendering system for large meshes. In *Proceedings of ACM SIGGRAPH 2000, Computer Graphics Proceedings*, pages 343–352, 2000.

[LRC$^+$02] D. Luebke, M. Reddy, J. Cohen, A. Varshney, B. Watson, and R. Huebner. *Level of Detail for 3D Graphics.* Morgan Kaufmann, 2002.

[LW85] M. Levoy and T. Whitted. The use of points as a display primitive. Technical report, University of North Carolina at Chapel Hill Technical Report TR 85-022, 1985.

[Ngu01] Baoquan Chen. Minh Xuan Nguyen. Pop:a hybrid point and polygon rendering system for large data. In *IEEE Visualization 2001*, pages 45–52, 2001.

[PZvBG00] H. Pfister, M. Zwicker, J. van Baar, and M. Gross. Surfels: Surface elements as rendering primitives. In *Proceedings of ACM SIGGRAPH 2000, Computer Graphics Proceedings*, pages 335–342, 2000.

[RW80] S.M. Rubin and T Whitted. A 3-dimensional representation for fast rendering of complex scenes. In *Proceedings of ACM SIGGRAPH 1980, Computer Graphics Proceedings*, 1980.

[Sta03] Carsten Dachsbacher. Christian Vogelgsang. Marc Stamminger. Sequential point trees. In *Proceedings of ACM SIGGRAPH 2003, Computer Graphics Proceedings*, pages 657–662, 2003.