2005

Master Texture Space: An Efficient Encoding for Projectively Mapped Objects

David Guinnip
University of Kentucky

ABSTRACT OF THESIS

David Guinnip

The Graduate School

University of Kentucky

2005

Master Texture Space: An Efficient Encoding for Projectively Mapped Objects

---

ABSTRACT OF THESIS

A thesis submitted in partial fulfillment of the
requirements for the degree of Master of Science in the
College of Engineering
at the University of Kentucky

By

David Guinnip

Lexington, Kentucky

Director: Dr. Grzegorz W. Wasilkowski, Associate Professor of Computer

Science

Lexington, Kentucky

2005

ABSTRACT OF THESIS

Master Texture Space: An Efficient Encoding for Projectively Mapped Objects

Abstract

Projectively textured models are used in an increasingly large number of applications that dynamically combine images with a simple geometric surface in a viewpoint dependent way. These models can provide visual fidelity while retaining the effects afforded by geometric approximation such as shadow casting and accurate perspective distortion. However, the number of stored views can be quite large and novel views must be synthesized during the rendering process because no single view may correctly texture the entire object surface. This work introduces the Master Texture encoding and demonstrates that the encoding increases the utility of projectively textured objects by reducing render-time operations. Encoding involves three steps; 1) all image regions that correspond to the same geometric mesh element are extracted and warped to a facet of uniform size and shape, 2) an efficient packing of these facets into a new Master Texture image is computed, and 3) the visibility of each pixel in the new Master Texture data is guaranteed using a simple algorithm to discard occluded pixels in each view. Because the encoding implicitly represents the multi-view geometry of the multiple images, a single texture mesh is sufficient to render the view-dependent model. More importantly, every Master Texture image can correctly texture the entire surface of the object, removing expensive computations such as visibility analysis from the rendering algorithm. A benefit of this encoding is the support for pixel-wise view synthesis. The utility of pixel-wise view synthesis is demonstrated with a real-time Master Texture encoded VDTM application. Pixel-wise synthesis is also demonstrated with an algorithm that distills a set of Master Texture images to a single view-independent Master Texture image.

KEYWORDS: View-dependent Projective Texture Mapping,Visiblity Analysis, View Synthesis, Triangle Packing, Rectangle Packing

Master Texture Space: An Efficient Encoding for Projectively Mapped Objects

By

David Guinnip

_____

Director of THESIS

_____

Director of Graduate Studies

_____

RULES FOR THE USE OF THESIS

Unpublished thesises submitted for the Master's degree and deposited in the University of Kentucky Library are as a rule open for inspection, but are to be used only with due regard to the rights of the authors. Bibliographical references may be noted, but quotations or summaries of parts may be published only with the permission of the author, and with the usual scholarly acknowledgements.

Extensive copying or publication of the thesis in whole or in part also requires the consent of the Dean of the Graduate School of the University of Kentucky.

A library that borrows this thesis for use by its patrons is expected to secure the signature of each user.

Name Date

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

MASTER THESIS

David Guinnip

The Graduate School

University of Kentucky

2005

Master Texture Space: An Efficient Encoding for Projectively Mapped Objects

---

THESIS

A thesis submitted in partial fulfillment of the
requirements for the degree of Master of Science in the
College of Engineering
at the University of Kentucky

By

David Guinnip

Lexington, Kentucky

Director: Dr. Grzegorz W. Wasilkowski, Associate Professor of Computer

Science

Lexington, Kentucky

2005

ACKNOWLEDGEMENTS

I would like to thank my adviser, Dr. Christopher Jaynes, for his helpful guidance, support, and encouragement throughout the completion of this thesis. I would like to thank Dr. Ruigang Yang and Dr. Etienne Grossman for giving me assistance and encouragement. I would like to thank my family for their love and support.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Image-based modeling is based on the observation that images can be used to render views of a scene without the need for an explicit representation of the underlying geometry. By eliminating or reducing the reliance on accurate geometry, image-based modeling has proven to be a powerful method to accurately render views of a scene [6, 8, 10, 11, 33]. Although all image-based methods represent a scene with a collection of images, techniques vary in the number of stored images, how these images are combined with scene geometry, and the rendering algorithms that produce viewpoints of the model not contained in image data.

View-dependent projective texture mapping (VDTM) has proven to be a particularly powerful image-based modeling technique for many domains and applications [5, 2, 10, 12, 27]. In contrast to other approaches that rely on image data alone, VDTM dynamically combines image-data with a model according to the current viewpoint of the scene. If calibration information (i.e. the intrinsic properties of the camera as well as its extrinsic pose with respect to the object) is known about each view, the perspective mapping between each view and the object geometry can be established and image pixels can be projected onto the object surfaces from each view. Because this approach utilizes model geometry, rendering effects such as shadow casting, perspective distortion, and self-occlusions are supported. At the same time, high-resolution data, constructed from one or more exemplar images of the real-world object, provides photo-realistic surface texture.

An unfortunate drawback to these methods however, is the large number of views required to accurately reproduce a scene. Often, as the complexity of the scene increases

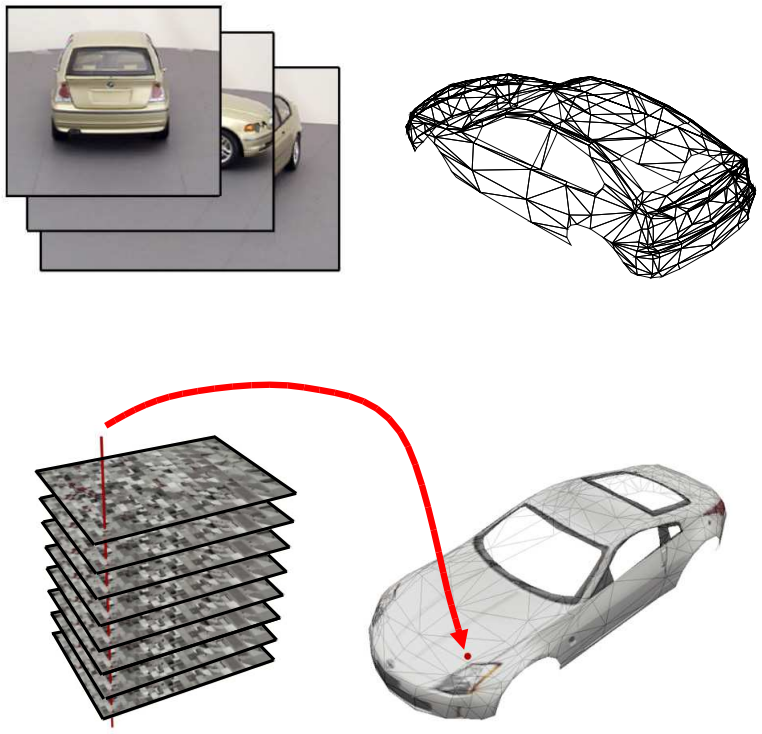


Figure 1.1: Traditional view-dependent texture mapping versus the Master Texture Space encoding. (Top) Traditional approach. Exemplar images must be stored with a different set of texture coordinates per image. As new views of the object are needed at run-time, these different texture coordinates must be available. (Bottom) After encoding, the Master Texture space hold a new set of images that require only a single set of texture coordinates. A single pixel (shown as a line) in all images corresponds to the same point on the object mesh.

either the mesh geometry must more accurately reflect the scene or the number of images must increase. As a result, researchers have begun to focus on compression, storage, and novel view generation techniques to alleviate these problems [5,11,13,14,31]. This work introduces an efficient representation and corresponding encoding process that was specifically designed for the image-based, view-dependent projective mapping domain. The technique reorganizes the image space so that a given pixel location across the image set corresponds to the same surface location. The result is a set of images that share the same texture coordinates regardless of viewpoint. This encoding improves the efficiency of image storage as the underlying projective geometry can be discarded and traditional image-based

operations such as view synthesis, view interpolation and editing become straightforward in the newly encoded image space [12]. This Master Texture Encoding reorders pixel data corresponding to mesh triangles by computing the optimal placement of texture triangles into a new 2D image plane. Optimal tiling of a plane using triangle elements is an NP-complete problem [4] and we introduce a new algorithm that searchers for an approximate solution via simulated annealing.

Once encoded, a given point on the object surface corresponds to the same pixel location in all views, regardless of the initial multi-view geometry to which the images correspond. This is significantly different from traditional view-dependent texture mapping scenarios in which the mapping from image to surface points is either implicitly encoded in a set of projection matrices or explicitly stored as a set of texture coordinates per-image. Figure 1.1 depicts the general difference behind a traditionally encoded VDTM model and the Master Texture representation. Pixel-wise alignment of images encoded in Master Texture space can be utilized to perform view synthesis directly in image space. Pixel-wise alignment allows Master Texture images to support view synthesis at render-time as well as off-line. Applications that utilize view-synthesis in the two desperate domains are presented in Chapter 5.

Chapter 2

Related Work

The Master Texture encoding is inspired by recent progress in combining geometric models with view-dependent texture information to produce realistic scenes at render time. Early image-based modeling and rendering techniques formalized the use of pixel data, potentially captured from real-world scenes, for rendering without explicit geometric descriptions [25, 29, 30, 33]. Pure image-based representations are beneficial for creating novel views when a large number of match points are recoverable and the entire scene can be represented in the set of exemplar images or some function thereof. Several techniques have been presented for performing view-synthesis in this domain [7, 15]. Light Field rendering [9, 16, 33], plenoptic function recovery [30, 20], Opacity Hull 3D photography [19], and novel view synthesis based on trilinear constraints [3] are focused on exploiting the image-based representation to generate novel views directly from the available images and, in some cases, camera calibration information. Hybrid methods that use both image data and a simple surface description of the scene have been shown to alleviate several of these problems [5, 10, 11, 14]. Exemplar images are projected onto a surface model from their known viewing location and then rendered from a novel viewpoint. In this way, perspective effects and occlusions are partially reproduced in the new view based on the accuracy of the underlying surface geometry. Research in VDTM mainly focuses on reducing the storage requirements in order to facilitate larger numbers of views and remove rendering artifacts. Debevec et al. introduced a view interpolation algorithm to synthesize novel views at render time by combining information from multiple exemplar images that are close (in terms of viewing parameters) to the desired view [11]. The approach pre-computes visibility

information for each mesh face and stores this information in a view-map data structure. At render time, the view-map is queried to correctly combine different exemplar images on a single mesh face to insure that texture information contains only visible pixels. The algorithm produces smoothly blended texture information from the exemplar views. As the work of [11] suggests, visibility analysis is an important aspect of accurate novel view generation. A caveat of projective texture mapping is the necessity to perform visibility analysis at render-time. Furthermore, complex processing of the existing mesh must occur in order to detect and eliminate special cases such as partially visible mesh elements as seen from any exemplar image view [34]. Such processing has the potential to increase the complexity of the underlying mesh. Alternatively, images encoded in a Master Texture space contain pixels that represent unique Euclidean positions on the surface mesh. Therefore, visibility analysis can be performed directly in the image space without altering the existing mesh. Novel view generation in Master Texture space is equivalent to linear interpolation of the Master Texture images. Furthermore, the set of Master Textures can be synthesized into a single Master Texture through direct image space operations, which extends the utility of the data set to rendering applications that do not support view-dependent texture mapping.

In work similar to our own, [22] describes an Eigen-Texture encoding scheme that stores all the views of a particular model face in a single image texture. This representation has the advantage of being amenable to off-line compression methods and results show that the technique is able to achieve between 5:1 and 15:1 compression rates with little or no loss in image fidelity. However, render-time texture compression is not addressed, and since an Eigen-Texture is constructed from a single mesh element under different illuminations, rendering an arbitrary view must access all Eigen-Textures [22]. Perhaps future research involving the Master Texture encoding could involve incorporating Eigen-Texture compression for off-line storage of the image data set, using the packing map (see section 3.2) to transfer between the Eigen-Texture space and the Master Texture space.

The Master Texture Encoding can be interpreted as an algorithm for automatic texture atlas generation (ATAG) [17, 18, 26]. The basic goal of existing ATAG algorithms is to

create a texture atlas that can be easily painted *a-posteriori* through a 3D Painting System without causing visual artifacts from segmentation or parameterization while maximizing packing efficiency [17]. Unlike the Master Texture encoding, previous methods for ATAG do not generally consider texture information as input, and do not base quality of the encoding on preserving source texture information. Since VDTM data sets provide color information *a-priori*, our algorithm only needs to parameterize a surface element in texture space to minimize the image-space difference of the encoded and exemplar elements. Also, image-based editing of Master Texture data sets can be performed prior to encoding [12], so the amenability for image-based editing of encoded data sets is not a concern. Therefore, the sole concern of segmentation and packing in the Master Texture algorithm is minimizing empty space in destination textures, resulting in packing efficiency ranging from 85 - 95% (see Chapter 4). The general Master Texture algorithm was presented in [12] and here we further develop along a number of lines including 1) a new and more efficient method for facet-packing, 2) a novel visibility analysis phase that exploits properties of the Master Texture space to discard unused image information, 3) a real-time rendering algorithm, and 4) a simple Master Texture synthesis algorithm to distill the set of Master Textures to a single, view-independent data set. The latter contribution significantly reduces the size of the data set and extends the usefulness Master Texture objects to rendering architectures that do not support view-dependent rendering. In addition, we explore the behavior of the algorithm under a wider variety of test cases and propose new directions for this and similar research. Results demonstrate that the new Master Texture space preserves image fidelity contained in the exemplar views, achieves reasonable compression, and facilitates efficient rendering and manipulation of the encoded images for a wide variety of rendering applications.

Chapter 3

Master Texture Encoding Algorithm

3.1 Introduction

The Master Texture encoding algorithm consists of three stages. The first stage parameterizes the local basis of each of the *m* triangular mesh elements in the *n* exemplar images. The result is a set of *n* face elements, or $facets$, for each of the *m* mesh elements, of uniform size, shape and orientation. The next stage of the algorithm determines a packing map that will determine an optimal packing for the set of *m* facets extracted and parameterized from each exemplar image *n*. A two-stage annealing process first pairs facets into rectangles that minimize wasted space across the pairings. Next, the set of $\lceil n/2 \rceil$ rectangles from the first annealing process are packed into a destination 2D plane that minimizes wasted space. The result of the second annealing phase is the packing map, which gives coordinates to place each of the *m* facets into the 2D plane of the n exemplar views - the Master Texture images. The final stage of the algorithm performs visibility analysis to fill occluded pixels in the Master Texture images set with visible pixels, resulting in a Master Texture image set that can accurately texture the surface mesh from any view-point, while maintaining view-dependent effects such as specular highlights and surface details represented in the view-dependent image data. Results show that the encoded data sets reduce render-time storage requirements by reducing texture dimensions and eliminating the need to compute or store multiple sets of texture coordinates. The explicit benefit of storage reduction legitimizes the use of the Master Texture encoding for VDTM in domains that involve resource constraints (i.e. transmission of a VDTM over a limited bandwidth link). However, an

Figure 3.1: Example of facet parameterization. (top row) Three images of a real-world car object. A single mesh face corresponding (insert on each view, top row) as seen in each image. (bottom row) Facets are parameterized to uniform shape, size, and orientation.

implicit benefit of the encoding is the correspondence of Euclidean positions on the surface mesh to pixels across the set of stored Master Textures. The Master Texture encoding guarantees that a pixel, $(i, j)_k$, in Master Texture image k corresponds to the same point on the surface as pixel (i, j) in all other Master Textures. *Pixelwise correspondence* is a beneficial property of the Master Texture encoding because view correspondence operations including view synthesis can occur on the now pixel-aligned data directly in image space.

3.2 Facet Parameterization

Given a set of n different exemplar images, each of the m mesh triangles are projected into all views to produce $n * m$ image facets. For a single mesh triangle, n facets, one for each view, are computed using the 3 triangular vertices and the n projection matrices corresponding to the views.

$$p_k^i = \mathbf{P}_i x_k, k = 1..3 \tag{3.1}$$

Where x_k is the three-dimensional homogeneous point of the triangle vertex k, p_k^i is the resulting two-dimensional homogenous image coordinates of the facet in image i, and $\mathbf{P}$ is the 4x3 projection matrix that describes how world points appear in exemplar view i.

After Equation 3.1 has been computed for all facets in the data set, all projection matrices are discarded and further operations now take place on the resulting n different 2D facets for that mesh triangle in image space. We refer to the set of facets p_k^i, corresponding to a single mesh triangle as a $facet\ family$, where i ranges from 1 to the number of views available. Figure 3.1 shows a model as seen from three different exemplar views. The three triangular facets, corresponding to the same mesh element (i.e. the same facet family) are highlighted on each exemplar view. Note that the facets are deformed due to perspective effects. Once extracted, the all the facets within each family are warped into a uniform size and shape. For a given family, each facet is transformed to a target right triangle of width w and height h whose longest axis is aligned with that of the x-axis in image space. For a given w and h, a two-dimensional affine warp, $\mathbf{X} = [a_{11}a_{12}a_{13}a_{21}a_{22}a_{23}]\mathbf{T}$, is derived from the three corresponding points on the initial triangle applied to each of the n facets:

$$\mathbf{A} = \begin{bmatrix} x_0 & y_0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_0 & y_0 & 1 \\ x_1 & y_1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_1 & y_1 & 1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_2 & y_2 & 1 \end{bmatrix}, \mathbf{B} = \begin{bmatrix} 0 \\ 0 \\ w \\ 0 \\ 0 \\ h \end{bmatrix}, \mathbf{A}x = \mathbf{B} \tag{3.2}$$

Where $\mathbf{A}$ is the set of initial image coordinates of the mesh triangle, $\mathbf{B}$ is a vector containing the corresponding target coordinates of the warped facet. Note that $\mathbf{B}$ is constructed so that the resulting facets are all aligned with the image axes. The elements of x are then determined from the least squares solution:

$$X = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{B} \tag{3.3}$$

As opposed to approaches that warp triangles to the maximum size of any triangle as seen in any view [12], we use a *mean enclosing triangle* approach, that computes the axis-aligned, right triangle that is minimally distant from all triangles in the family. This increases the compression achieved by the algorithm while avoiding undue sampling artifacts by guaranteeing that the target triangle width, w, and height, h, are close to the facet family under consideration. This is accomplished by minimizing the sum of the distances between facet vertices before and after warping:

$$\arg\min w,h \sum_{j=1}^{k} \sum_{i=1}^{3} \|D(x_i, \mathbf{A}_{wh} x_i)\| \tag{3.4}$$

Where *D(a,b)* is the Euclidean distance between point x_i and the point warped by the affine transform $\mathbf{A}_{wh}x_i$. Note that $\mathbf{A}_{wh}$ is the affine transform matrix given by Equations 3.2 and 3.3 for particular *w* and *h* values. This way the size of a packing triangle for a facet family remains fixed and resampling must only occur once. Once the facet families have been parameterized, each facet family F_n is analyzed to see if it is of uniform color across all views. If the image difference across F_n is less than an intensity difference threshold , and the facets are of uniform color, then the dimension of the facet family can be reduced to fixed dimension *d*. To eliminate sampling artifacts during render-time rasterization, we used fixed *d* of 5. Our results used an intensity difference threshold of 5.

3.3 Efficient Facet Packing

At this stage of processing, each family contains *n* facets of the same size and shape even though they have been derived from *n* unique views. Each of these facets are placed into *n* different Master Texture images at the same position and orientation to guarantee alignment of the facets across views. For a mesh of size *m*, one facet from each of the *m* facet families is placed into the same Master Texture. Facets are restricted to 90-degree rotations in order to minimize aliasing artifacts. Once facets have been packed, the result is a new Mater Texture image for each of the original exemplar views that contain one warped facet from each of the original facet families. The packing algorithm maximizes pixel use in the Master Texture image while at the same time minimizing the size of the image required. The result of the packing algorithm for one view, referred to as a packing map, is applied directly to all the remaining views to guarantee that each image in the Master Texture space is pixel-wise aligned. The packing process first pairs the set of *n* right triangles in rectangular bounding regions so that the total area of all rectangles is minimized. This step is motivated by the desire to reduce the triangle-packing problem to one of efficient tiling using rectangular elements. Although research has produced interesting theoretical

results related to tiling triangles on the plane, more efficient techniques focus on the more constrained problem of placing rectangles on a planar surface to maximize coverage while allowing only translation and 90-degree rotations of the individual rectangles [21]. An optimal packing of the rectangles within the smallest possible image region is then computed in order to derive a final Master Texture image. Specifically, given the set of $\lceil n/2 \rceil$ rectangles, the smallest bounding rectangle (image) that minimizes the number of points within that rectangle not covered by image data is computed. Efficient packing of polygon elements on the plane is not a new research topic and has been studied with varying constraints as the two-dimensional tiling problem. In similar work, Soucy et al. generates an image to texture-map the object surface by packing textured triangles into the 2D image plane [31]. The new triangle-packing algorithm described here is based on an iterative optimization process rather than a set of packing heuristics and is able to outperform (in terms of space utilization) the method described in [31]]. Optimal solutions for specific 2D tiling packing problems have been motivated by industrial applications, such as stock cutting, data storage, and VLSI design [21], and we draw upon these to develop a solution to our problem.

3.3.1 Triangle Pairing

Facets are first converted into rectangles by pairing similar triangles and fitting a bounding rectangle to the result. Care must be taken at this stage to efficiently pair triangles so as to efficiently make use of the resulting rectangular region. Since the facets are right triangular and the base and height run along scan lines, two facets can be paired by transposing one of the facets along the x and y-axis, and placing it at the bottom right coordinate of the minimum bounding rectangular region of the two facets. To distinguish between the facets in a pairing, the transposed facet is called the flipped facet, while the other will be referred to as the base facet. Equation 3.5 determines the size of the resulting bounding rectangle that encloses the paired facets.

$$\arg\min((\arg\max(k,n) * \arg\max(l,m)), (\arg\max(l,n) * \arg\max(k,m))) \qquad (3.5)$$

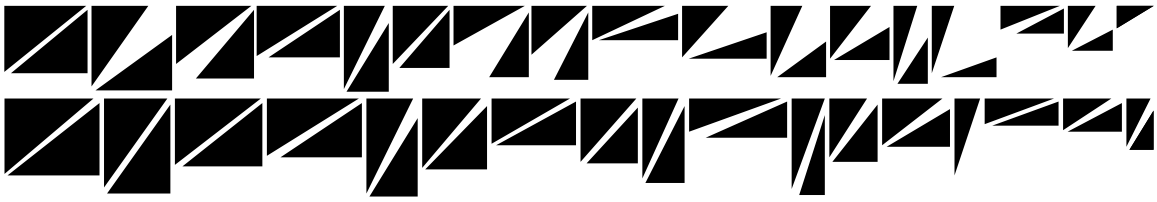


Figure 3.2: A section of the triangle-pairing for the BMW Max-Area-Facet data set. The top image shows the initial pairings using simple heuristics to select pairings for search initialization. It has a pairing efficiency (measured as total area used within the rectangular bounding boxes)of 81%. The bottom image is after optimization, using the 5th cooling schedule (Figure 4.2). The final pairing efficiency is 97%.

Where (m,n) and (k,l) are the dimensions of the un-flipped facet and flipped facet, respectively. The problem of pairing right triangles to produce a set of rectangles with a globally minimum area is a specific instance of the 1-D bin-packing problem: given an unlimited number of variable sized bins and a fixed set of items, place items within bins in such a way as to minimize the total bin size required. Flipped triangles play the role of the items while base triangles act as the bins. A packed bin then, is a base facet that has been paired with a flipped facet. The total size of a bin is determined by Equation 3.5, and unlike the general bin-packing problem, the bin size changes according to which item is contained in the bin. However, the goal remains identical: discover an optimal packing of the bins (base facets) with items (flipped facets) such that the total bin area is minimized.

Initially, the data set is conditioned by pairing the largest two un-paired facets until all facets have either been paired, or in the case of an odd number of facets, the smallest facet remains the sole un-paired facet. A search process iterates over potential parings using the total coverage area as an error metric. The space of possible pairing is search via an annealing process to avoid local minima in the search space. Three operations that occur with equal probability are used to move through the search space: a facet may be flipped (if paired, its partner is also flipped), two items may exchange bins, or two bins may exchange items. Varying cooling schedules were used and results show-pairing efficiency ranging from 86% to 97%. Figure 3.2 shows a set of triangular facets extracted from a single view of the BMW model. Initial pairings are shown at top and the result after the annealing process terminates is shown in the bottom image of Figure 3.2. Although the pair-selection

algorithm is iterative, we have found that in practice it outperforms more straightforward heuristic-based methods such as directly pairing triangles according to the similarity of their hypotenuse length. It should also be noted that this is an off-line process and, because facets are of the same size/shape across all images, the pairing selections in one image determine the same pairing for all images. Results related to pairing efficiency using a variety of cooling schedules are shown in Chapter 4.

3.3.2 Rectangle Tiling

The set of rectangles resulting from the pairing assignments computed in the previous phase of processing are now efficiently packed into a bounding rectangular region. There has been much success in applying simulated annealing to solve the rectangle-packing problem in other application domains such as circuit board design [13, 21]. This success is, in part, due to the sequence-pair representation of the search space. This representation supports flexible operations during the annealing process and efficient computation of the cost function for an arbitrary configuration in the search space [21]. The representation encodes a particular packing into two sequences (X,Y) that describe a complete ordering of the rectangles on each of the two axes of the plane. A full description of the sequence-pair is out of the scope of this paper, but a brief overview follows (for further details, refer to [21]).

A sequence-pair is a pair of sequences (X,Y) of *n* elements representing a list of *n* rectangles. Geometric constraints on the sequence describe the geometric locations of the rectangles in the packing. Rather than explicitly represent the actual 2*D* offset of each rectangle in the plane, the order of appearance in a sequence describes whether a given rectangle *x* is *"above"*, *"below"*, *"to the right of"*, or *"to the left of"* another rectangle $\hat{x}$ in the packed plane. For instance, if $\hat{x}$ is before *x* in both sequences, then $\hat{x}$ is *"to the left of"* and *"below" x* in the packed image plane. Because the sequence-pair representation does not explicitly represent 2D coordinates on the plane, blocks cannot overlap one another. In this way, a valid packing is intrinsic to the sequence-pair representation and checking for degenerative configurations (such as overlapping rectangles) while perturbing the sequence-pair is not necessary. Therefore a packing can be perturbed by simply

shuffling the entries in either of the two sequences.

In order to measure the quality of a particular sequence-pair, it is necessary to convert the representation to actual rectangle coordinates so that the total size of the bounding region and coverage can be measured. Quality is given by the ratio of the sum of the component rectangle areas over the total bounding rectangle size. Since the sequence-pair will be evaluated for quality potentially thousands of times during the annealing process, it is important that this measure can be computed quickly. In practice, the bounding rectangle size is directly derived from the sequence-pair using a fast technique, called the longest common subsequence evaluation [32]. This method has been shown to run 60 times faster than earlier graph-based methods [21] and is used here to compute the quality of a particular sequence-pair during the annealing process. For further details on sequence-pair evaluation via longest common subsequences, the reader is directed to [32].

3.3.3 Simulated Annealing Search for Efficient Rectangle Packing

Simulated annealing search framework is used to compute the optimal rectangle packing over the sequence-pair representation space. To improve the time required to converge to a solution, rectangle placement is initialized using straightforward heuristics. First, rectangles are flipped so the largest dimension is the width, then the set of rectangles are sorted by width from largest to smallest. In previous work [12], the largest rectangle to the smallest was simply packed in order into the new image under the assumption that packing efficiency will decrease if large rectangles are left to be packed into nearly full tiling. This requires that a fixed image width for the Master Texture packing is determined *a-priori*, which is set to be the square root of the total sum of the areas of the rectangles to be packed. The sequence-pair is similarly initialized through the following operations. The Y sequence is constructed by adding each element from the X sequence to the Y sequence from back-to-front until the sum of the widths of the rectangles in the Y sequence fits within the approximate dimension and no more rectangles can be added. This process is repeated until all rectangles from X have been added to the Y sequence.

Figure 3.3 shows an example of the initial conditioning for a set of ten rectangles. This

Rectangle ID	width	height
1	23	24
2	15	16
3	9	9
4	12	13
5	26	26
6	50	10
7	20	20
8	55	30
9	10	20
10	100	10

(10	8	6	5	1	7	9	2	4	3)
(x	x	x	x	x	x	x	x	x	x)
(x	x	x	x	x	x	x	x	x	10)
(x	x	x	x	x	x	x	x	8	10)
(x	x	x	x	x	x	5	6	8	10)
(x	x	x	1	7	2	5	6	8	10)
(4	9	3	1	7	2	5	6	8	10)

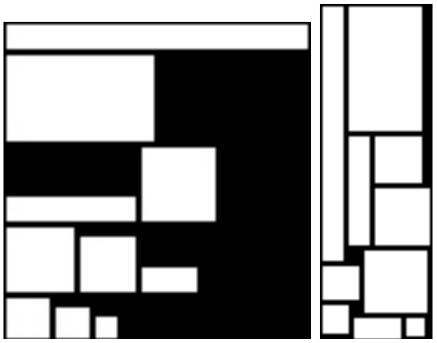


Figure 3.3: Packing example using the sequence-pair. (Far Left) A set of 10 rectangles. (Middle left) Conditioning the initial sequence-pair set. Rows show the sequence-pair at each iteration of the conditioning algorithm. (Middle right) The conditioned packing of the sequence-pair that is the initial state of the annealing process. This initial packing has dimension 100 x 104 with 50% efficiency. (Far right) The result of the packing after the annealing process has dimension 43 x 130 with 94% efficiency.

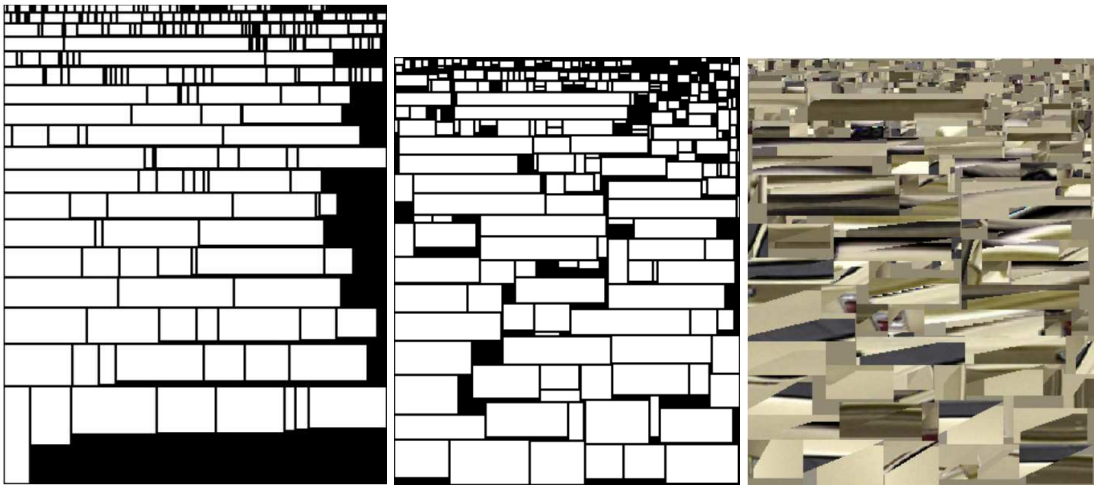


Figure 3.4: Example of the iterative facet packing process. (Left) Initial estimate of image size and packing (see Text). (Middle) Final result after 22,433 iterations of the annealing process using schedule 5 (See Figure 4.2). In this example, packing efficiency (measured as ratio of covered pixels to total pixels in image) was increased from 89.3% to 93.3%. (Right) Final Master Texture image containing the facet pixel data.

sequence-pair initial guess is then optimized via simulated annealing. Four operations are used in the annealing process: 1) Transpose a rectangle (rotates the rectangle 90-degrees in the plane), 2) Swap two elements from the X sequence, 3) Swap two elements from the Y sequence, 4) Swap two elements from both sequences. Varying cooling schedules were used and results show packing efficiency, measured as total area covered by pixels in the final image, ranging from 85% to 98% (see Section 4). When the process terminates, a packing map that determines how all facets are placed into the Master Texture has been discovered. Figure 3.4 shows this iterative packing process for a family of rectangles on the BMW data set and the resulting Master Texture. Once computed, this same packing map is then applied to all other facet sets to derive the complete Master Texture space.

3.4 Visibility Determination

The goal of the visibility analysis phase of the encoding algorithm is to classify every pixel in the Master Texture images as either visible or occluded. Once this determination has been made, occluded pixels can be directly replaced with visible pixels from other views. Because the same pixel corresponds to the same point on the object surface, an occluded pixel (i,j) in Master Texture image k need only look for a replacement in pixel (i,j) of the remaining Master Texture image set. Initially, the entire underlying mesh is rendered from a given exemplar viewing position and its depth buffer values are stored. This is similar to other visibility methods that assume the presence of a model or simple geometric primitives [23, 28]. Next, the weighted Euclidean position for every pixel used to texture map mesh faces is derived from the barycentric coordinate weights of a given pixel that are applied to each Euclidean position of the mesh face containing that pixel. Given these weights, the resulting position of a pixel $l_{x,y,z}$, is given by Equation 3.6.

$$l_{x,y,z} = m_{x,y,z}w_m + n_{x,y,z}w_n + o_{x,y,z}w_o \tag{3.6}$$

In this equation,$m_{x,y,z}$,$n_{x,y,z}$, and $o_{x,y,z}$ are the Euclidean positions of the mesh face vertices containing the pixel, while w_m, w_n, and w_o are barycentric weights of the relative positions

Figure 3.5: (Top)Exemplar image from the ”truck” data set. Triangle outlined in white on cab illustrates a partially occluded surface facet from this view. (Far left) Partially occluded facet extracted and warped. (Middle left) Pixels determined to be occluded shown in white. (Middle right) Closest neighbor facet that contains visible pixels. (Far right) Synthesized facet containing only visible data.

for each position respectively. Pixels that are not used to texture map faces (i.e. pixels taking up wasted space in the packed images) are marked invalid and are discarded for the remainder of the algorithm. To determine if a Euclidean position on the model surface is visible from a given view, the projection of the mesh point onto the image plane is computed along with the depth buffer value at that image point. If the depth buffer value of the extracted pixel is the same as the depth buffer values in the stored depth buffer, then the pixel is visible from the corresponding viewing position; otherwise it is occluded. This process is used to classify the visibility of each Master Texture image pixel in the set of Master Texture images. Once classified, occluded pixels are then filled with valid pixel information. This is a straightforward operation due to the pixel-wise alignment property of the Master Texture space. If a pixel (i, j) is occluded in view p, then it is filled with a visible pixel (i, j) in the Master image t closest to view p. The dot product of the optic axis of view p and each of the Master Texture images in which pixel (i, j) is visible is

computed. The view with the largest dot product contains the closest visible pixel, and the entry in Master image *p* is replaced with this value. At the conclusion of visibility synthesis, all Master images are now filled with visible pixel values. Figure 3.5 shows this process for a particular facet on the ”truck” model.

Chapter 4

Results

The Master Texture encoding algorithm was run on several data sets in order to study algorithm behavior under a variety of conditions. Results are discussed in terms of compression rates and fidelity loss with respect to the original data. Computer graphics models were used for two of the four models (Lamp and Helicopter) so that ground truth data was readily available. A third and fourth model (BMW automobile and Ford F150 truck) were created from a low-polygon count model combined with real-world images of a scale model captured under controlled conditions. Data sets of varying mesh complexity and varying numbers of exemplar images were used. The criteria for evaluating the approach were 1) efficiency of rendering a Master Textured object from changing viewpoints, 2) compression rate of the Master Texture encoding as compared to traditional texture mapping, 3) triangle packing efficiency, and 4) fidelity of the Master Texture encoded model after rendering as compared to the original data set. The algorithm was run on all data sets using an Intel 2.4Ghz machine.

The four data sets are shown in Table 4.1. Master texture and mesh samples from the data sets are shown in Figure 4.1. Encoding times are largely dependent on mesh

Model	Mesh Faces	Views	Exemplar Image Size
Truck	1801	28	512x512
Lamp	1842	90	640x480
Helicopter	436	360	720x576
BMW	651	28	640x480

Table 4.1: Exemplar data set statistics.

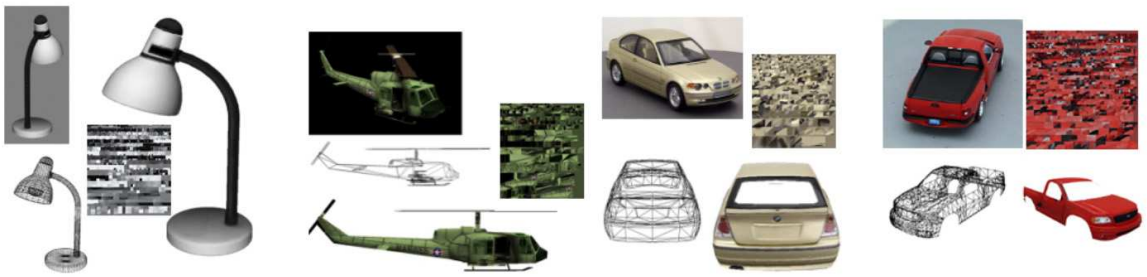
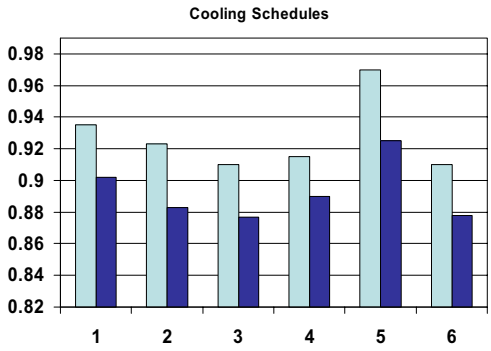


Figure 4.1: Results of the Master Texture encoding applied to the four different test objects. Each example shows an exemplar image (top left), object mesh (bottom left), example Master Texture, and the rendered view (at right).

resolution and number of iterations performed in the annealing process. Encoding times typically range from a few seconds, for a 651 polygon model containing 28 views using a rapid cooling schedule, to 86 minutes for 1842 polygon model with 90 views using a slow cooling schedule. The pairing process was capable of producing efficient triangle pairs and demonstrated consistent pixel utilization of the bounding rectangle of 86 to 95%. These rectangular regions were then packed using the algorithm described in Section 3.3. Overall compression rates are greatly influenced by the resulting Master Texture image size as well as the efficiency of packing the rectangular regions into that image. For data sets with a relatively small numbers of triangular mesh elements, the annealing process provides the greatest increase in efficiency. For instance, the helicopter data set increased 7 percent for the mean and greatest enclosing encodings. In contrast, data sets with a larger



Cooling Schedule	Starting Temperature	Reduction Value
1	200	0.001
2	100	0.001
3	20	0.01
4	10	0.01
5	10	0.0001
6	0	0

Figure 4.2: A study of cooling rates on two different data sets. (Left) A plot of the total packing efficiency for the BMW and Helicopter data sets for each of the different cooling schedules. (Right) Cooling schedule includes the starting temperature (randomness) of the search and a reduction value that is applied at each iteration to dictate how rapidly the temperature is decreased.

number of triangular mesh elements receive a smaller increase in efficiency. The truck data set is a good example of this behavior with less than 1 percent increase in efficiency in mean and greatest enclosing encodings. This effect is partly due to the fact that our initialization heuristics are more efficient for the larger mesh element data sets. Of course, the particular cooling schedule utilized in the annealing process will influence both the time to convergence and the efficiency of the rectangle packing in the Master Texture image. Figure 4.2 depicts the effect of overall packing efficiency in the Master Texture image versus six different cooling schedules labelled 1 through 6 on the BMW data set. The cooling schedules differ in the starting temperature and the rate at which that temperature is iteratively lowered.

Figure 4.2 (Right) shows the different parameters for each of the six different cooling schedules. Compression rates of the three different objects were measured. Compression rates were measured as the ratio of total bytes required for traditional VDTM versus VTDM using the Master Texture encoding. Table 4.2 summarizes the results of the experiment using both the maximum enclosing triangle (Max MT) and mean enclosing triangle (Mean MT) warping methods during the encoding process. Compression rates are related to the size of exemplar views, the percentage of pixels in those views that related to the model, and even the distribution of those views around the model itself. In general, compression rates are reasonable for data sets with a large number of views and a moderate number of polygons on the object mesh. Table 4.3 clearly demonstrates the advantage of using the mean enclosing triangle warping approach for compression.

Artifacts may arise from the facet-warping phase when image data is resampled accord-

Model	Max MT Size	Comp. Ratio	Sched.	Time (minutes)	Mean MT Size	Comp. Ratio
Truck	622x589	0.84	1	72	342x325	2.77
Lamp	448x417	1.63	1	86	236x226	5.88
Helicopter	663x546	1.14	5	14	271x228	6.66
BMW	904x785	0.48	5	49	490x420	1.42

Table 4.2: Compression rates for four different data sets of varying mesh size, exemplar image size, and number of views.

Model	Max % diff.	M.I.D.(Max MT)	Mean % diff.	M.I.D. (Mean MT)
Truck	2.16	2.3	2.66	2.6
Lamp	5.86	4.6	6.16	4.6
Helicopter	5.86	4.6	6.06	5.0
BMW	5.03	4.0	5.63	4.3

Table 4.3: Image difference scores of the four data sets using both maximum enclosing triangle and mean triangle warping

Figure 4.3: A side-by-side comparison of the BMW model rendered from a fixed view. (a) Rendering using traditional view-dependent texture mapping and an exemplar view. (b) Rendering from the same view using a Master Texture encoded image. (c) Direct image difference of first two renderings. Differences have been normalized to a range of 0-255 for visualization purposes. Only 5.6% of the 640x480 pixels in the image are different with a mean intensity difference of less than five.

ing to the new facet shape prior to packing into the Master Texture space. In order to study the effects of the encoding process on fidelity, Master Texture encoded images are used to re-render the model from the same view as an exemplar image. A direct difference of the two images, then, reveals how and where the Master Texture encoding result differs from the ground truth data. Table 4.3 shows the results of this experiment with our data sets using both the maximum enclosing and mean target triangle warping methods (see Section 3.2). Figure 4.3 shows an original exemplar image of the BMW model and the difference images that result from re-rendering the scene using Master Texture encodings and computing an absolute difference. Figure 4.3(middle) corresponds to a Master Texture image that was computed using the mean enclosing triangle method and Figure 4.3(right) corresponds to the image difference of the first two images. The method used to select a target triangle influences the amount of warp (and potential resampling) that may be present in the data. Unsurprisingly, Table 4.3 shows that the maximum enclosing triangle approach produces images that are closer to the original data but do not achieve the same compression

rates. Whatever the warping method used, the Master Texture images are very similar in appearance to their original counterparts. Across the five different test data sets, the mean percentage of pixels that were different in re-rendered views was less than five percent and the mean intensity difference of those pixels was less than five intensity levels.

Chapter 5

Applications

Intrinsic in the properties of Master Texture-encoded objects is the support for simple pixel-wise view synthesis. We show the utility of view synthesis in two distinct domains. First, a real-time application for displaying a Master Texture encoded object using today's commodity graphics hardware is introduced. Next, an algorithm for compressing the set of Master images to a single Master image is demonstrated. The resulting Master Texture data can then be efficiently transmitted across the web. An interactive viewing client-server application was developed in order to demonstrate this capability.

5.1 Real-time View-dependent Application

A real-time model visualization tool that makes use of the view-dependent Master Texture encoding was developed to demonstrate the advantages of working within the Master Texture space. The application is loosely based on the system described in [11], and is capable of render-time blending of a VDTM object. The input to the application consists of the geometric mesh, set of Master encoded images I(views), and the optic axis vectors corresponding to the source view of each I. To ensure smooth blending as the viewing position changes, we utilize a view map data structure [11]. We chose this technique over a more robust technique, such as the one presented in [5], because of its speed and amenability to a single-pass hardware-accelerated implementation. The limitation is that it assumes an outside looking in model of the viewing space, and all source views point roughly at the view map origin.

A conditioning step assigns the closest source view in the polar coordinate grid to each of the nodes of the view map. The algorithm proceeds by dynamically choosing the three closest views given the current viewing position and each is assigned a blending weight. The current viewing position p_v is first mapped into polar coordinate grid and intersecting triangle t of p_v is determined. The three nodes of t become the three closest reference views.

Blending weights are then computed as the barycentric coordinates of p_v on t. The weight W_i for a given source view i of t is given by

$$W_i = A_i / A_t \tag{5.1}$$

where A_i is the area of the triangle formed by p_v and the two vertices other than i, and A_t represents the total area of t. Blending guarantees source view weights will vary smoothly as the viewpoint changes [11]. It is important to note blending effectiveness is related to the ratio of the number of source views to the number of nodes in the view map. As the ratio approaches zero, the three node indices for any given viewing position will reference the same view. As the ratio approaches infinity, source views will not be indexed in the view map. Empirically, a ratio of 1/6 yields good results and was used for the application.

Once blending weights have been determined, the final color of each fragment f_c is determined by applying the blending weights to the corresponding textures using the following equation,

$$f_c = \sum_{i=1}^{3} c_i * W_i \tag{5.2}$$

where c_i is the texture color and W_i is the normalized weight. Because data exists in a Master Texture space, color assignments can be applied on a per-fragment basis in a pixel shader. Figure 5.1 shows the source code for the pixel shader. It is written in NVIDIA's CG language [24]. This is the only operation performed by the fragment program, since more complicated operations such as visibility analysis are now unnecessary. It is also important to note that a single set of texture coordinates are loaded and used for all Master images. This eliminates the overhead of texture coordinate computations and dynamic loading of three independent texture coordinate sets per frame. Our application contains a mirror

```
struct fragment{
                    float4 position  : POSITION;
                    float4 color0    : COLOR0;
                    float2 texcoord0 : TEXCOORD0;
};
struct pixel{
                    float4 color     : COLOR;
};
pixel main (fragment IN,    const uniform sampler2D texA,
                                        const uniform sampler2D texB,
                                        const uniform sampler2D texC,
                                        const uniform float1 weightA,
                                        const uniform float1 weightB,
                                        const uniform float1 weightC){
                    pixel OUT;
                    //Fetch the RGB texel color from the textures
                    float4 texelColorA = tex2D(texA, IN.texcoord0);//all 3 use same texcoord!
                    float4 texelColorB = tex2D(texB, IN.texcoord0);//all 3 use same texcoord!
                    float4 texelColorC = tex2D(texC, IN.texcoord0);//all 3 use same texcoord!
                    OUT.color=float4(           texelColorA.r * weightA +
                                                texelColorA.g * weightA +
                                                texelColorA.b * weightA,
                                                texelColorB.r * weightB +
                                                texelColorB.g * weightB +
                                                texelColorB.b* weightB,
                                                texelColorC.r * weightC +
                                                texelColorC.g * weightC +
                                                texelColorC.b * weightC);
                    return OUT;
}
```

Figure 5.1: NVIDIA pixel shader for performing per-fragment texture blending.

toggle that can enable a ”mirror” that shows the reflection of the data set. Figure 11 left shows a screen shot of the application with the mirror toggle enabled. The mirror effect is simply a second rendering pass from a view 180-degrees around the object. The second rendering pass uses the same views and weights as the first render pass, eliminating the need to perform a second query of the view map. This illustrates how any single Master Texture, or combination thereof, can accurately texture the entire object surface.

The real-time application generates a rendering rate of 85 fps for the BMW data set with mirror enabled and 75 fps with mirror disabled. We tested the application on a Dell

Precision 480 with an Intel Xeon processor and an NVIDIA Quadro FX 500 graphics card.

5.2 Off-line View Synthesis

Off-line view synthesis through direct image-space operations is a benefit exclusive to Master Texture encoded objects. Image-based edits in one Master Texture image, for example, can be directly propagated to all views without the need to recapture or modify all images [12]. Here we introduce a simple algorithm that exploits this property to derive a single Master Texture image from a set of Master views that correctly textures the object from any view. The algorithm results in a data set that is drastically smaller (in terms of file size) and view-independent, allowing greater access to the encoded object.

5.2.1 View-independent Synthesis Algorithm

The view synthesis algorithm for Master Texture encoded VDTM models consists of filling an initially empty Master Texture with pixel information that most accurately represents the model from any given viewing position. The result is a single Master image that can be utilized to texture map the underlying surface in a view-independent fashion. This is quite simple since occluded pixels do not exist in the Master image space at the conclusion of the encoding, therefore reducing the complexity of the algorithm to simple geometric relationships of the pixellocations and their relation to the Euclidean positions of the enclosing facet vertices.

The algorithm proceeds as follows: for each vertex of a facet in the Master image, find the closest view to the vertex. A closest view associated with a facet vertex F is determined by finding the largest dot product produced by multiplying each exemplar view position vector by F's vertex normal $\overrightarrow{N_f}$. Next, for each pixel p contained in the empty facet, store the weighted sum (in terms of barycentric coordinate weights of the pixel location relative to the container facet, see Equation 5.1 and Equation 5.2) of the color components of the corresponding p in the three closest views. Barycentric coordinates are used for blending to guarantee continuity of color values across face boundaries. Once the Master image is generated, the object can be viewed by statically texturing the view-independent Master

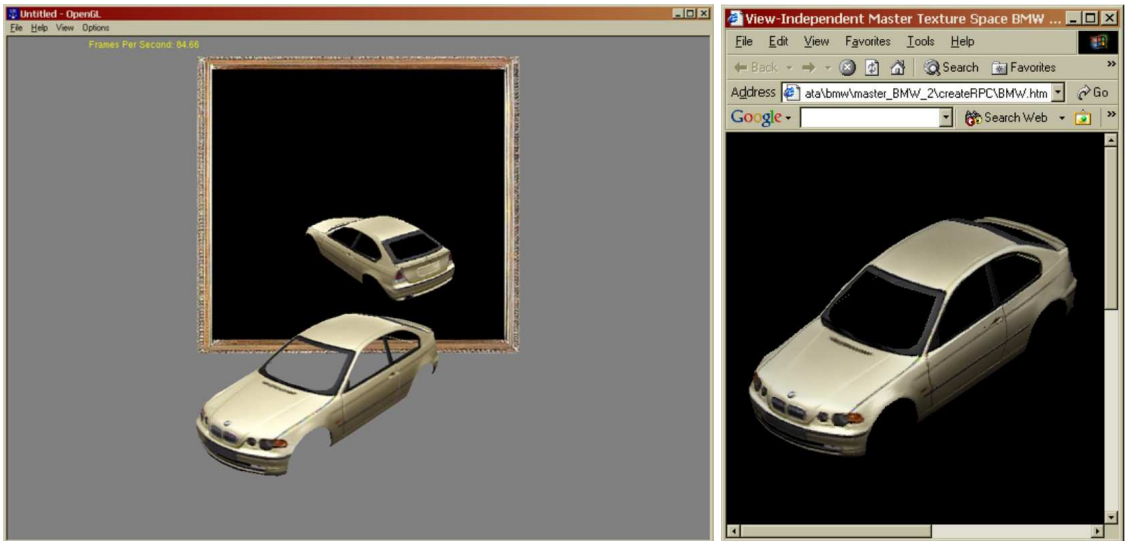


Figure 5.2: Screen shots from Master Texture encoding applications. Right image is a screen shot of the VDTM application with mirror toggle enabled. Left image is a screen shot of the View-Independent Master Texture encoded object as seen from a web page with the ArchVision RPC ActiveX control [1].

image onto the underlying geometry via the texture coordinate set. Although dynamic effects such as lighting are lossed, compression of the original data set is drastic. For the data sets discussed in Chapter 4, for example, compression was increased by a factor of the total source views for each data set. These smaller data sets, although possibly not as visually stunning, are much more accessible to the spectrum real-time 3D applications, such as web-based viewers. To demonstrate the utility of the view-independent algorithm, we have converted several Master Texture data sets to the standard ArchVision RPC format [2]. These data sets can be downloaded and viewed through the web via the ArchVision ActiveX Viewer [1] at http://www.metaverselab.org/research/imb/mastertexture/index.html.

Chapter 6

Conclusion

We have introduced an efficient representation and corresponding encoding process that specifically addresses problems related to image-based, view-dependent projective rendering. The technique achieves reasonable compression of the image-data and supports efficient operations in the newly compressed space that are typically applied to image-based objects such as view synthesis, view interpolation and editing. The Master Texture space is an object-specific encoding that compresses the set of texture meshes, computed by multiplying the underlying mesh by the set of projection matrices, into a single global texture mesh. Exemplar images are then uniformly transformed to adhere to the new texture mesh. This single texture mesh and the set of newly encoded Master Textures are sufficient to reconstruct the view-dependently textured object from arbitrary views. Once complete, all pixel data in the Master Textures is valid for view interpolation and other image-based operations. The Master Texture encoding avoids storing (or computing) multiple sets of textures with the use of a single, and fixed, set of texture coordinates. Furthermore, because all views of the object have been transformed into the Master Texture space, any point on the object surface corresponds to the same pixel in all images.

Because visibility information is explicitly encoded in the Master Texture space, subsequent visibility analysis and interpolation operations are straightforward linear operations on the pixel-wise aligned data. Perhaps more importantly, the visibility information encoded in the Master Texture space can be used to easily synthesize views. The utility of image synthesis in Master Texture space was demonstrated in two distinct domains: a real-time VDTM rendering application using todays commodity graphics hardware, and

an algorithm to produce a single view-independent Master Texture object of drastically reduced file size for use in applications that do not support VDTM, e.g. the ArchVision RPC ActiveX Viewer. We are currently in the process of exploring new ways to exploit the Master Texture space to increase efficiency of traditional image-based rendering techniques. One promising area of research is the automatic reduction of complex computer graphics models to a Master Texture encoded view-dependent representation. For example, by re-rendering a complex scene under different lighting conditions and encoding the illumination changes as different Master Textures, subsequence relighting of the model can be accomplished using simple pixel-wise addition in the Master Texture space. Another area of research is a progressive encoding of the Master Texture space for resolution control for different application domains.

Bibliography

[1] ArchVision. RPC activex viewer. www.archvision.com/rpcactivexviewer, 2005.

[2] ArchVision. RPC technology. www.archvision.com, 2005.

[3] S. Avidan and A. Shashua. Novel view synthesis by cascading trilinear tensors. *IEEE Transactions On Visualization and Computer Graphics*, 4(4):293–306, 1998.

[4] B.S. Baker, E. G. Coffman, and R. L. Rivest. Orthogonal packings in two dimensions. *SIAM Journal of Computing*, 9(4):846–855, 1980.

[5] C. Buehler, M. Bosse, L. McMillan, S. J. Gortler, and M. F. Cohen. Unstructured lumigraph rendering. In *SIGGRAPH 2001, Computer Graphics Proceedings*, pages 425–432, 2001.

[6] D. Burschka, D. Cobzas, Z. Dodds, G. Hager, M. Jagersand, and K. Yerex. Recent methods for image-based modeling and rendering. In *IEEE Virtual Reality Tutorial 1*, March 2003.

[7] C.S. Carpenter, B. Seales, C. Jaynes, and R. Stevens. Automated basis-view and match-point selection for the archvision rpc image-based model. In *Proceedings of the International Conference on Multimedia*, pages 577–581, September 2001.

[8] S. E. Chen and L. Williams. View interpolation for image synthesis. *Computer Graphics*, 27(Annual Conference Series):279–288, 1993.

[9] W-C. Chen, J-Y. Bouguet, M. Chu, , and R. Grzesuk. Light field mapping: Efficient representation and hardware rendering of surface light fields. In *SIGGRAPH 2002*, pages 447–456, 2002.

[10] P. Debevec, C. J. Taylor, and J. Malik. Modeling and rendering architecture from photographs: A hybrid geometry and image-based approach. *Computer Graphics*, 30(Annual Conference Series):11–20, 1996.

[11] P. Debevec, Y. Yu, and G. Boshukov. Efficient view-dependent image-based rendering with projective texture-mapping. In *Eurogaphics Rendering Workshop*, pages 105–116, June 1998.

[12] D. Guinnip, C. Jaynes, D. Rice, and R. Stevens. Efficient image-based projective mapping using the Master Texture Encoding. In *Winter School Conference in Computer Graphics*, pages 49–56, 2003.

[13] S. Kirkpatrick, C.D. Gelatt, , and M.P. Vecchi. Optimisation by simulated annealing. *Science*, 220:671–680, 1983.

[14] C. S. Kurashima, R. Yang, and A. Lastra. Combining approximate geometry with view-dependent texture mapping - a hybrid approach to 3d video teleconferencing. In *SIGGRAPI*, pages 112–120, October 2002.

[15] S. Laveau and O. Faugeras. 3-d scene representation as a collection of images. In *Proceedings of 12th International Conference on Pattern Recognition*, volume 1, pages 689–691, 1994.

[16] M. Levoy and P. Hannrahan. Light field rendering. In *SIGGRAPH 1996*, pages 31–42, 1996.

[17] B. Levy, S. Petitjean, N. Rayand, and J.Maillot. Least squares conformal maps for automatic texture atlas generation. In *SIGGRAPH 2002*, pages 362–371, 2002.

[18] J. Maillot, H.Yahia, and A. Verroust. Interactive texture mapping. In *SIGGRAPH 1993*, pages 27–34, 1993.

[19] W. Matusik, H. Pfister, A. Ngan, P. Beardsley, R. Ziegler, and L. McMillan. Image-based 3d photography using opacity hulls. In *SIGGRAPH 2002*, pages 427–437, 2002.

[20] L. McMillan and G. Bishop. Plenoptic modeling: An image-based rendering system. In *SIGGRAPH 1995*, pages 39–46, 1995.

[21] H. Murata, K. Fujiyoshi, S. Nakatake, and Y. Kajitani. Vlsi module placement based on rectangle-packing by the sequence pair. *IEEE Transaction on Computer Aided Design of Integrated Circuits and Systems*, 15(12):1518–1524, 1996.

[22] K. Nishino, Y. Sato, and K. Ikeuchi. Eigen-texture method: Appearance compression and synthesis based on a 3d model. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(10):1257–1265, 2001.

[23] NVIDIA. Hardware shadow mapping. developer.nvidia.com/object/hwshadowmap_paper.html.

[24] NVIDIA. Nvidia cg toolkit. developer.nvidia.com/object/cg_toolkit.html.

[25] K. Pulli, M. Cohen, T. Duchamp, H. Hoppe, L. Shapiro, and W. Stuetzle. View-based rendering: Visualizing real objects from scanned range and color data. In *Proceedings of 8th Eurographics Workshop on Rendering*, pages 22–34, June 1997.

[26] N. Ray and B. Levy. Hierarchical least squares conformal maps. In *11th Pacific Conference on Computer Graphics and Applications 2003*, pages 263–270, 2003.

[27] REALVIZ. developer/producer of image-processing software and applications, 2005.

[28] M. Segal, C. Korobkin, R. Widenfelt, J. Foran, and P. Haeberli. Fast shadows and lighting effects using texture mapping. In *SIGGRAPH 1992*, pages 249–252, July 1992.

[29] S. Seitz and C.R. Dyer. View morphing. In *SIGGRAPH 1996*, pages 21–30, 1996.

[30] S. Seitz and K. Kutulakos. Plenoptic image editing. In *Proc. 6th International Conference in Computer Vision (ICCV '98)*, pages 17–24, 1998.

[31] M. Soucy, G. Godin, and M. Rioux. A texture-mapping approach for the compression of colored 3d triangulations. *The Visual Computer*, 12:503–514, 1996.

[32] X. Tang, R. Tian, and D. F. Wong. Fast evaluation of sequence pair in block placement by longest common subsequence computation. In *Conference on Design, Automation and Test in Europe*, pages 106–111, 2000.

[33] D. Wood, D. Azuma, K. Aldinger, B. Curless, T. Duchamp, D. Salesin, and W. Stuetzle. Surface light fields for 3d photography. In *SIGGRAPH 2000*, pages 287–296, 2000.

[34] Y. Yu. Efficient visibility processing for projective texture-mapping. *Journal of Computers and Graphics*, 23(2):245–253, 1999.

Vita

Name: David Guinnip
Date of Birth: Aug. 29th, 1977
Place of Birth: Lexington, Kentucky

Education

- B.S. in Computer Science and Studio Art, Transylvania University, Lexington, KY, May, 2000.

Publications

- David Guinnip, Christopher Jaynes, David Rice, and Randall Stevens. ”Efficient Image-Based Projective Mapping using the Master Texture Space Encoding”. In *Winter School Conference of Computer Graphics*, pages 49-56, 2003.
- David Guinnip, Ruigang Yang, and Liang Wang. ”View-Dependent Textured Splatting for Rendering Live Scenes” Techincal Report 431-05, Department of Computer Science, University of Kentucky, Lexington, KY, 2005.

Presentations

- David Guinnip and Ruigang Yang. ”View-Dependent Textured Splatting for Rendering Live Scenes” SIGGRAPH Poster Presentation, 2004.

Honors and Awards

- Finalist in ACM Student Research Competition for SIGGRAPH 2004 Poster, ”View-Dependent Textured Splatting for Rendering Live Scenes”.
- Teaching Assistantship, University of Kentucky, 2000-2001.
- Research Assistantship, University of Kentucky, 2000-2001.
- Research Assistantship, University of Kentucky, 2003-2004.