Content Aware Texture Compression

Yu-Cheng Chiu
Department of Computer Science
National Chiao Tung University
HsinChu, Taiwan
22kinds@gmail.com

Yu-Shuen Wang
Department of Computer Science
National Chiao Tung University
HsinChu Taiwan
yushuen@cs.nctu.edu.tw

***Abstract*—Efficient and high quality texture compression is important because computational powers are always limited, especially for embedded systems such as mobile phones. To access random texels instantly, down sampling and S3TC are the most commonly used methods due to the fixed compression ratio at every local region. However, the methods are content oblivious, which uniformly discard texel information. Therefore, we present content aware warp to reduce a texture resolution, where homogeneous regions are squeezed more to retain more structural information. We then relocate texture coordinates of the 3D model as the way of relocating texels, such that rendering the model with our compressed texture requires no additional decompression cost. Our texture compression technique can cooperate with existing methods such as S3TC since the compression strategies are independent. By reducing texture resolution in a content aware manner, followed by texture compression using S3TC, as the experiments show, we achieve higher compression ratios without degrading visual quality significantly.**

***Keywords*-Texture compression, Atlas, Content aware warp**

I. Introduction

Texture compression long has been used in computer graphics to offer low cost but high quality rendering results. Although computational powers are rapidly increasing nowadays, reducing graphics memory and bandwidth consumption without degrading texture quality significantly is always important. Particularly, this technique is commonly seen in embedded systems such as mobile phones because their electronic powers are limited. While texture decompression is required by each texture access, general image compression methods such as JPEG and H.264 are not adequate in this scenario because their decompression processes are not simple and efficient, even though the methods achieves both high quality and high compression ratio simultaneously.

We present a content aware texture compression approach that requires no decompression when rendering a model with our compressed textures. It can work along or cooperate with existing S3TC methods. The idea is to warp the texture, where homogeneous regions are squeezed, to reduce the texture resolution. Since texels in homogeneous regions are similar, storing them using fewer samples and followed by up sampling does not introduce noticeable visual artifacts (Figure 1). Thus, we strive to take advantages from them in order to save memory and bandwidth consumption of texture access. To implement this idea, we partition each texture chart using a triangular mesh and warp the mesh to relocate texels. Specifically, an importance value of each triangle is first computed by averaging the gradient magnitudes of interior texels. Each triangle is then prevented from squeezing according to its importance value when the texture resolution is reduced. Since regions with high gradient texels are less squeezed, more samples will be stored in the reduced texture and more details will be retained. In contrast, texels in homogeneous regions are discarded to reduce memory consumption.

Given that homogeneous regions may be small or surrounded by high-gradient texels, squeezing only homogeneous regions is not able to effectively reduce the texture resolution. Fortunately, we observed that interpolation along the direction that has the minimal color variation introduces the least blurring artifacts because the texels used for interpolation have similar colors. We thus compute an optimal scaling direction of each triangle based on its interior texel gradients and encourage the triangle to be squeezed along this direction in order to avoid visual blurs in case homogeneous regions are run out. To implement this idea, we compress the area of each texture chart belong to a 3D model and pack them to a 2D texture as tight as possible. Considering the texture charts are compressed and placed at different positions, we update the mapping between texels and primitives so that each primitive can fetch the right content when rendering models. That is, each texture coordinate is embedded in the triangular mesh and it can be trilinearly interpolated using the surrounding vertex positions. Given that the mapping between texels and primitives is still retained, our method requires no decompression process when rendering texture models.

We demonstrate the effectiveness of our technique by comparing it to down sampling, S3TC, and signal specialized parameterization [1]. Experiments show that our compressed textures suffer from less stretching artifacts due to flexible compression ratios and content aware strategy. Note that our algorithm is not introduced to replace existing bit-encoding methods such as S3TC, but to provide a strategy that can cooperate with them. We show that a

Figure 1. We apply content aware warp to compress a texture to 40% of the original size, in which homogeneous regions are squeezed while structural regions are retained (left). Texture coordinates of the 3D model are then relocated to match the texel relocations so that each primitive can be correctly rendered. We compare the 3D model rendered with the original and the compressed textures (right). As can be seen, blurring artifacts are not noticeable although the texture is compressed.

texture can be warped to a smaller resolution, followed by applying S3TC compression, to further reduce memory and bandwidth consumption.

II. Related Work

Texture compression long has been studied for several decades. Different from traditional image compression techniques, rendering a compressed texture requires fast decoding and random access of texels because the decompression is frequently preceded and how a renderer will access a texture is unknown in advance. Therefore, the compression methods such as JPEG and H.264 are not adequate even though they can achieve both high quality and high compression ratios simultaneously.

Indexed color is the first presented texture compression method [2], which also is the first method considered for hardware implementation [3]. This method compresses the texture into a color table containing a number of colors and an index for each texel that maps to the stored color. Vector quantization extends the indexed color approach from each texel to 2×2 or 4×4 texel blocks in order to minimize the error. Although these methods are good at compressing textures that have few colors, they are no longer used because they require an indirection table lookup.

S3TC [4], [5] was introduced and became the most common texture compression system and available on all consumer-level hardwares. S3TC contains five formats (i.e., DXT1 to DXT5) that are different in the handing of alpha channel. S3TC partitions a texture into 4×4 texel blocks. Each block has two 16 bit RGB565 colors and a control code per texel used for linear interpolation. The fixed compression ratio enables S3TC to randomly access texels and its linear interpolation also prevents discontinuity artifacts that are commonly seen in the indexed color method. Later, Nvidia extended S3TC to VTC [6] that allows the width and the height of a texture other than multiples of 4. ETC [7], [8], [9] was presented to extend S3TC based on the idea that humans are more sensitive to changes in luminance than in chrominance. Hence, in their system, several texels may share one chrominance but each of them has its own luminance. There are also some texture compression methods optimized for two-channel textures [10], [11] and normal maps [12] extended from S3TC. We refer the readers to [13] for more details. Although the above methods are fast and enjoy simple implementation in hardware, they are of a fixed compression ratio and are oblivious to texture content. Hence, complex textures may be over-compressed while homogeneous textures are potentially under-compressed.

There have been methods presented to parameterize a 3D mesh onto a 2D plane while allocating more texture space in regions with greater signal detail [14], [1]. Both the previous works and our method have the same goal but handle the meshes in different conditions. Specifically, the previous methods require the signals obtained from the 3D mesh such as scanned color data or normal maps. Although signals in 3D could be projected from a 2D texture, the demanded texture mapping results in resampling distortion.

Besides, their stretch parameterization assumes signals to be piecewise linear interpolant of vertex attributes and the metric tensor to be constant or linear over the triangle. Although the authors applied 1-to-4 subdivisions to triangles to increase the mesh resolution, the strategy may still neglect small features and blur the texture again. Overall, we believe the signal specialized parameterization is good at allocating texture samples for 3D signals but the additional resampling distortion is inevitable if signals are projected from 2D textures. In contrast, our texture compression focuses on 2D throughout the process. It is faithful to the original texture samples when reducing the texture size.

Our idea comes from the image compression based on selected data pruning [15]. The method first reduces the resolution of an image by removing rows and columns that have low color gradients. Then, it applies H.264 to compress and decompress the reduced image. The final reconstruction is based on high-order edge-directed interpolation. Although effective, the interpolation demands heavy computational cost and can be used only at image compression. Both this and our methods take advantages from homogeneous regions, which are similar to image retargeting [16], [17], [18], [19], [20], [21] but have different objectives.

III. System overview

Most 3D meshes are partitioned into several planner-approximated charts when they are parameterized to 2D space for texture mapping. Therefore, our system compresses each 2D chart individually and repacks them to compress the texture. Specifically, we partition the chart using a triangular mesh and warp the mesh to a smaller size (Figure 2). Our goal is to reduce the texture resolution while retaining as many high-gradient texels as possible. Namely, triangles covering homogeneous texels are squeezed more while the others are retained during our content aware warp. To implement this idea, we formulate the requirements into a number of energy terms and solve an optimization problem to obtain the warped chart. Next, we pack the warped charts into a smaller rectangular space and compute the texel colors in the reduced texture using trilinear interpolation. Finally, by relocating texture coordinate of each vertex as the way of relocating texels, the mappings between texels and primitives are retained so that the texture model can be rendered correctly. We emphasize that rendering a model with this reduced texture requires no additional cost, although the compression may take some time.

We warp the texture of each chart by reducing either the width or the height iteratively to achieve compression. In each step, the texture is decompressed by inverse warp and then compared to its original version to measure the compression quality. Namely, root mean square error (RMS) is applied to measure the difference of texel colors caused by the compression. In our implementation, we set the reduction amount to be five texel size in each iteration and apply the greedy algorithm to reduce the texture size in the dimension that has a smaller increasing RMS error. This process repeats until the reconstruction error is larger than a threshold or the desired resolution is reached. Although the strategy takes many iterations, the overall computational cost is acceptable because our warping technique is fast and the computation of RMS value can be sped up by parallel computing.

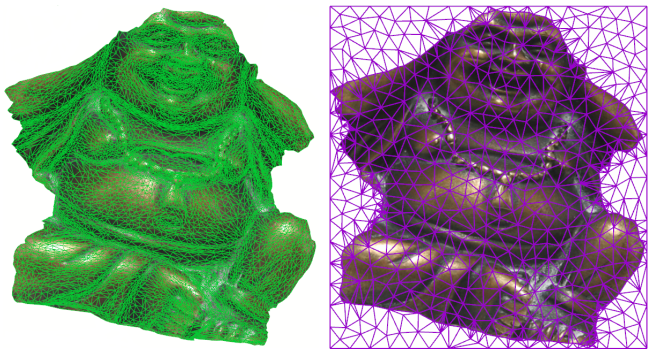

Figure 2. (left) The texture chart of a complex 3D model. (right) Our triangular mesh. We determine a coarser triangular mesh, where the boundary edges and vertices are retained, to speed up the compression procedure.

Once all the texture charts are compressed, we adopt a heuristic method to pack them into a 2D texture as tight as possible so as to save the space. Given that the final texture resolution is unknown in advance, we first set the texture to have zero space and iteratively grow its size as the charts are sequentially packed.

IV. Content aware texture warp

We apply Delaunay triangulation to partition the chart, with each triangle covering less than 500 texels and each angle larger than 20 degrees. To warp the chart, we compute the bounding box of the chart and set the four corners as constrained vertices when triangulation (Figure 2). Besides, the boundary vertices and edges of the chart are also constrained in this step so that the texture coordinates in the chart can be enclosed after the content aware warp. Our system guarantees that all triangles cover valid texels after the compression if the warp does not introduce edge flipping.

Let $\mathbf{M} = (\mathbf{V}, \mathbf{F})$ be the triangular mesh used for the content aware warp, where $\mathbf{V} = \{\mathbf{v}_0, \mathbf{v}_1, ..., \mathbf{v}_m\}$, $\mathbf{v} \in \mathcal{R}^2$ is the vertex position and $\mathbf{F}$ is the set of triangles. Also let $\mathbf{U} = \{\mathbf{u}_0, \mathbf{u}_1, ..., \mathbf{u}_n\}$, $\mathbf{u} \in \mathcal{R}^2$ be the texture coordinates of a chart that are updated after $\mathbf{M}$ is warped. For clarity, we call the triangles formed by texture coordinates as primitives. Below we show only the warping of a chart and this approach is applied to all charts individually.

A. Importance value

We compute an importance value of each triangle $f \in \mathbf{F}$ by averaging the gradient magnitudes of interior texels. This value is then normalized to among 0.05 and 1.0, and

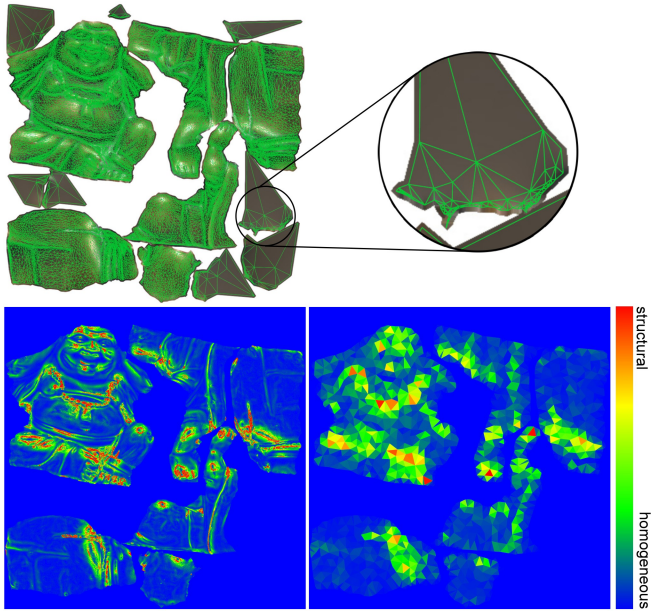

Figure 3. Since some texels are unused when rendering 3D models (top), they are neglected when computing gradient magnitudes (bottom left). Importance value of each triangle (bottom right) is then determined by averaging the interior gradient magnitudes. In these two maps, regions from homogeneous to structural are visualized from blue to green, yellow and red.

used to weight the energy terms described later. We set the smallest importance value to 0.05 in order to avoid numerical instability. As a texture may contain high level information such as faces, the face detection technique [22] is also applied to determine the importance of a triangle. Apparently, triangles with high importance values usually contain features and will retain more texels in the reduced texture to achieve high quality rendering result. Note that we set the texels that are not mapped to any primitive of the texture model to have zero gradient magnitudes because they are unused, as illustrated in Figure 3.

B. Optimal scaling direction

In addition to preserving triangles with larger importance values from squeezing, we encourage each triangle to compress in the direction that has the least color variation. The linear upsampling along this direction results in smaller blurring artifacts because the colors of the sampled texels are similar (Figure 4). To achieve this, for each texel i that is covered by triangle f, we consider its eight adjacent neighbors $j \in \mathbf{N}\{i\}$ and compute a set of color variation vectors using $\mathbf{h}_{ij} = d_{ij} \times \mathbf{e}_{ij}$, where d_{ij} is the norm of RGB color difference between neighboring texels, and $\mathbf{e}_{ij}$ denotes the normalized direction from texel i to texel j. As illustrated in Figure 5, the length of each variation vector implies how much blur that scaling along it would introduce. We thus compute the optimal scaling direction of each triangle by

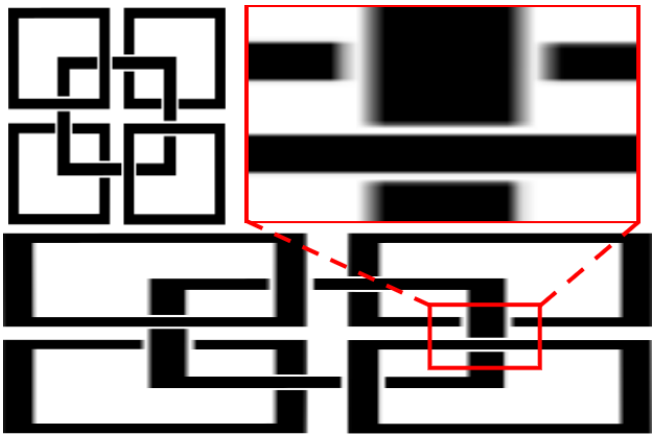

Figure 4. (top left) The original texture. The horizontally scaled texture (bottom). Blurring artifacts occur at vertical edges (top right) due to the large color variations in the horizontal direction.

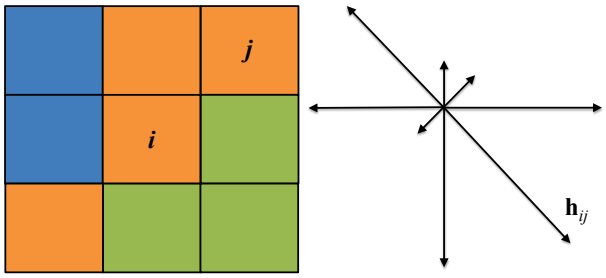

Figure 5. (left) The center texel i and its eight adjacent neighbors j. (right) The set of variation vectors $\mathbf{h}_{ij}$ are determined based on the color variations between texel i and j. We compute the optimal scaling direction that has the least color contrast so that the up sampling along this direction results in the minimal blurring artifacts.

searching a vector $\mathbf{A}_f$ that can minimize

$$\sum_{i \triangle f} \sum_{j \in \mathbf{N}(i)} |\mathbf{A}_f \cdot \mathbf{h}_{ij}|^2, \tag{1}$$

where $i \triangle f$ denotes that pixel i is covered by f. Intuitively, by considering each vector $\mathbf{h}_{ij}$ as a 2D point, this optimization problem can be solved using principal component analysis. While the major axis approximates the positions of these 2D points, which implies the largest color variation, we set our optimal scaling direction $\mathbf{A}_f$ to the minor axis to minimize upsampling blurring artifacts.

C. Constrained optimization

Our system scales down the texture while retaining as many high-gradient texels as possible by solving a constrained optimization problem. The objective function contains the energy terms that measure triangle squeeze, deviation of vertex collinearity, triangle fold over, and boundary straightness. Below we show the details of the presented energy terms.

Squeezing. We retain each triangle according to its importance value when the texture resolution is reduced. Considering homogeneous regions may be small, we also allow each triangle to be scaled along its optimal scaling direction. Considering that the triangle is scaled under an arbitrary axis, we first rotate the triangle to align this axis with a vertical or a horizontal line, then scale the triangle, and then rotate it back to its original orientation. Therefore, let θ_f be the angle between the horizontal axis and the optimal scaling direction of triangle f, we constrain triangle edges by minimizing the energy term

$$\Omega = \sum_{f\in\mathbf{F}} \sum_{\{i,j\}\in f} \omega_f \left|(\hat{\mathbf{v}}_i - \hat{\mathbf{v}}_j) - \mathbf{S}_f(\mathbf{v}_i - \mathbf{v}_j)\right|^2,$$
$$\text{where} \quad \mathbf{S}_f = \mathbf{R}_{\theta_f} \begin{bmatrix} s_f & 0 \\ 0 & 1 \end{bmatrix} \mathbf{R}_{\theta_f}^{-1}, \qquad \mathbf{R}_{\theta_f} = \begin{bmatrix} \cos\theta_f & -\sin\theta_f \\ \sin\theta_f & \cos\theta_f \end{bmatrix}, \tag{2}$$

ω_f is the importance value of f, $\hat{\mathbf{v}}_i \in \hat{\mathbf{V}}$ is the warped vertex position and s_f denotes the unknown scaling factor. Note that scaling each triangle along our determined optimal direction does not guarantee to be free from distortions. The larger $|\mathbf{A}_f \cdot \mathbf{h}_{ij}|$ implies the more noticeable artifacts even though the triangle is squeezed along $\mathbf{A}_f$. We thus enforce each scaling factor s_f to satisfy

$$s_f \geq \min(\frac{1}{\eta}\max|\mathbf{A}_f \cdot \mathbf{h}_{ij}|, 1), \tag{3}$$

where $\mathbf{h}_{ij}$ is a color variation vector. We set $\eta = 100$ in all our experimental results. Note that the largest $|\mathbf{A}_f \cdot \mathbf{h}_{ij}|$ is $\sqrt{3\times 255^2}$ in the definition of RGB color space.

Collinear constraint. We observed that a long edge of a texture primitive may be partitioned into several segments in order to prevent large triangles when computing Delaunay triangulation. Therefore, the segments should remain collinear after our content aware warp. Otherwise, the large primitive will cover invalid texels, as illustrated in Figure 6, because the deformation of a coarse primitive is not able to capture our non-linear texel relocations. Let E_ℓ be the segments partitioned from the edge of a texture primitive ℓ and $\mathbf{n}_\ell$ be the edge normal. To achieve the aim, we maintain their collinearity by constraining

$$(\hat{\mathbf{v}}_i - \hat{\mathbf{v}}_j)\cdot \mathbf{n}_\ell = 0, \qquad \forall\{i,j\} \in E_\ell. \tag{4}$$

Rectangular and fold-over constraints. We reduce the chart size by compressing its bounding rectangle. Hence, during the width or the height reduction of mesh $\mathbf{M}$, its regular shape should be retained and the vertices are prevented from moving across the four straight boundaries. To achieve a rectangular shape, we enforce the boundary vertices moving only along their respective boundary lines

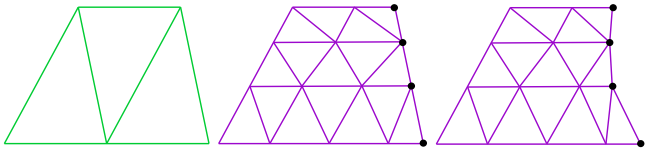

Figure 6. (left) Primitives of a texture chart. (middle) Our generated mesh may be denser than the original chart due to the constraints of size and angle. (right) Invalid texels may be mapped to the texture primitives if the black vertices do not remain collinear after our content aware warp.

during the warp. That is, we introduce the boundary constraint

$$\begin{cases} \hat{\mathbf{v}}_{i,x} = 0 & \text{if } \hat{\mathbf{v}}_{i,x} \text{ is on the left boundary} \\ \hat{\mathbf{v}}_{i,x} = D_w & \text{if } \hat{\mathbf{v}}_{i,x} \text{ on the right boundary} \\ \hat{\mathbf{v}}_{i,y} = 0 & \text{if } \hat{\mathbf{v}}_{i,y} \text{ is on the top boundary} \\ \hat{\mathbf{v}}_{i,y} = D_h & \text{if } \hat{\mathbf{v}}_{i,y} \text{ on the bottom boundary,} \end{cases} \tag{5}$$

where D_w and D_h are the reduced width and height of the bounding space, respectively. In addition, considering least squares optimization only approximates the given constraints, triangles may flip when they are squeezed, which results in vertices crossing the bounding space and triangles fetching wrong texels after compression. We thus enforce the distance between each vertex and its opposite edge of the triangle to be longer than a threshold to prevent triangles from squeezing into zero sizes or even flip. Formally, we expect each deformed vertex $\hat{\mathbf{v}}_i$ to satisfy

$$|\hat{\mathbf{v}}_i - \hat{\mathbf{p}}_{jk}| \geq \varepsilon, \tag{6}$$

where i, j, k are the three vertex indexes of a triangle, $\mathbf{p}_{jk} = \delta\hat{\mathbf{v}}_j + (1-\delta)\hat{\mathbf{v}}_k$ is the closest point to $\hat{\mathbf{v}}_i$ and δ is the combination weight computed from point-line theory. We set $\varepsilon = 3$ in all our experimental results. Under this setting, our system can guarantee at least some texels in each triangle are stored in the reduced texture even the interior content is homogeneous.

Optimization. We minimize the energy term Ω subject to the constraints described in Equations 2 - 6 to achieve a content aware texture warp. By setting the first derivative of the objective function to zero, we solve for the warped vertex positions $\hat{\mathbf{V}}$ and the correlated scaling factor of each triangle s_f using a linear system. Given that $\hat{\mathbf{V}}$ and s_f are unknown and correlated, the optimization can only be achieved by iterative updating. Specifically, we first set $s_f = 1$ as an initial guess and solve for $\hat{\mathbf{V}}$. To update s_f, we then rotate both the warped and the original triangles f using $\mathbf{R}_{\theta_f}$, followed by determining an affine transformation $\mathbf{T}_f$ that can transform the original triangle to the warped triangle, and set s_f to the top-left element of $\mathbf{T}_f$. Note that we set $s_f = \min(\frac{1}{\eta}\max|\mathbf{A}_f\cdot\mathbf{h}_{ij}|, 1)$ if the computed scaling factor is too small in order to prevent over squeezing. We repeat this iterative procedure until the system converges.

We point out that Equation 6 is an inequality constraint, which is inactive in the beginning of our texture warp.

0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
5	4	3	2	1	0	0	0	0
0	0	5	4	3	2	1	0	0
0	6	5	4	3	2	1	0	0
6	5	4	3	2	1	0	0	0
5	4	3	2	1	0	0	0	0
7	6	5	4	3	2	1	0	0
0	6	5	4	3	2	1	0	0

Figure 7. (left) We partition the texture chart using a uniform grid mesh with each grid containing 4×4 texels before the chart packing. (right) The encoded right-shift value indicates the number of skipped grids horizontally that is free of overlapping.

During the iterative procedure, we check if there is any vertex being too close to its opposite edge whenever $\hat{\mathbf{V}}$ is updated. If $|\hat{\mathbf{v}}_i - \hat{\mathbf{p}}_{jk}| < \varepsilon$, we add an additional constraint

$$\hat{\mathbf{v}}_i - \delta\hat{\mathbf{v}}_j - (1-\delta)\hat{\mathbf{v}}_k = \frac{\varepsilon}{|\hat{\mathbf{p}}_{jk}|}\hat{\mathbf{p}}_{jk}, \tag{7}$$

to the system and solve for the new vertex positions $\hat{\mathbf{V}}$, where $\hat{\mathbf{p}}_{jk}$ is obtained from the previous iteration.

V. Chart packing

Once all the texture charts are compressed, they are packed onto a 2D texture as tight as possible to save the texture resolution. Given that each chart could have various positions and orientations, this problem has been proof as NP-complete [23]. Fortunately, it has been well studied in industrial applications. We thus borrow the idea and simply modify the method to fit our requirements.

Considering a texture is consisted of discrete texels, we adopt a heuristic algorithm presented by Babu and Babu [24] to determine the chart position. Specifically, this method partitions the packing texture to 2D uniform grids, with each grid containing $n \times n$ pixels ($n = 4$ in our experimental results). It aims to place the chart possibly close to the bottom left point while iteratively updating the chart position rightward and upward until the collision is free. To speedup the process, the method encodes the right-shift value at each grid to indicate the horizontal movement of the new chart that is free of collision. The wide step prevents the frequent collision checks when the chart has to move in the horizontal direction. As for the update in the vertical direction, only a grid size is taken. Figure 7 shows the illustration. We also refer the readers to the encoding of right-shift value to [24] for more details.

Note that our packing requirement is different to that of [24]. In an industry application, the packing space is limited and clearly defined, and the number of charts is assumed infinite. However, our goal is to pack all the compressed charts within a smallest rectangular. We thus assume the size of the packing texture is unknown and iteratively grow it as the charts are packed sequentially. That is, let w and h be the width and the height of a packing space, where their initial values are both 0. Given a chart, our system checks whether or not the chart can be packed using the method of [24]. We also allow each chart to rotate by $\frac{\pi}{2}$, π, and $\frac{3\pi}{2}$ degrees to fit the empty area of the texture space. This discrete rotation does not require texel resampling and prevents quality reduction. If the packing space is too small to pack a chart, we extend either the width or the height of the space, which introduces the smallest additional space, to pack the chart. This process packs the charts in the order of chart area (i.e., from the largest to the smallest) and repeats until all the charts are packed. Although this greedy algorithm does not guarantee the packing space is minimum, we experientially found the strategy works well.

VI. Results and Discussions

We have implemented and tested our algorithm on a desktop PC with core i7 3.0 GHz CPU. We use the Cholesky solver to minimize Ω subject to a number of hard and inequality constraints (Equations 2 - 6) when warping meshes. After that, trilinear interpolation is applied to compute each texel color in the reduced texture based on the original and the warped meshes. Although our texture compression requires solving an optimization problem, the texture can be compressed once and then stored in the disk, where the re-compression whenever loading a texture to the GPU is not necessary. More importantly, rendering 3D models with our compressed textures requires no additional cost because texture coordinates are coherently relocated as well.

The computational cost of our system depends on the compression ratio and the density of the triangular mesh. The larger compression ratio results in more iterations. The denser triangular mesh indicates the more unknown variables are solved in each warp. Generally, compressing a texture with 1024×1024 resolution to about 30% of the original size takes around 3 minutes using our unoptimized code. We believe the performance could be greatly enhanced when RMS errors are computed in parallel.

We compared our method to down sampling and show the results in Figures 1 and 8 to demonstrate the effectiveness of our technique. Given that down sampling is oblivious to texture content, rendering 3D models with these reduced textures suffers from noticeable stretching artifacts. We also compared our method to [1] using D3DXUVAtlas functions implemented in Direct3D and show the results in Figure 9. Both methods retain important texels to compress textures and achieve high quality. We point out that our result contains less stretching artifacts when the model is close to the camera.

Our algorithm is flexible to not only the compression ratio but also the quality of a texture. Because texture charts are compressed individually, the system allows users to specify the importance of each chart so that meaningful regions such

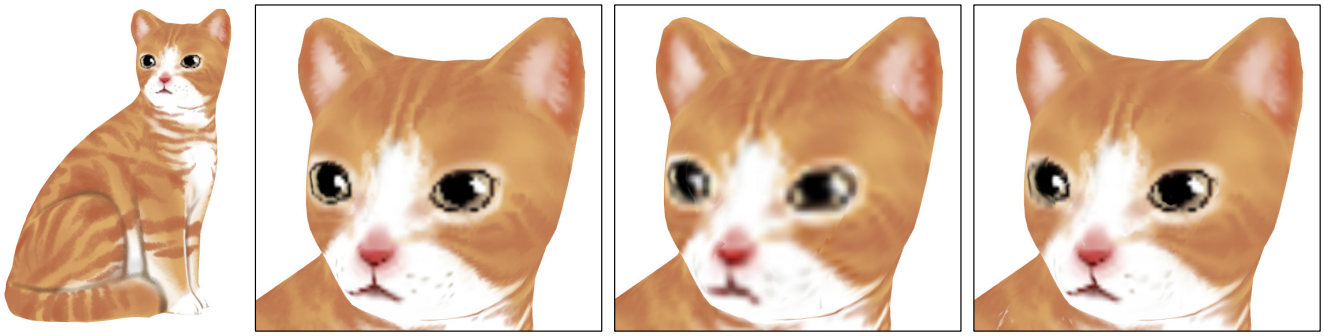

Figure 8. From left to right are the cat model, the cat's head rendered with the original, down sampled, and our compressed textures. Both the compressed textures have 57% of the original size. Compared to the down sampled texture, our result suffers from less aliasing artifacts when the camera is very close to the cat's face.

Figure 9. We render the Buddha model with the compressed textures that are obtained from [1] (left) and our method (right). Both textures are compressed to 24% of the original size. The Buddha's eyes are zoomed in and shown in top. As can be seen in the highlighted square regions, our result suffers from less stretching artifacts.

as faces could remain the original quality after compression. Figure 10 shows the result with and without user specified chart importance. In addition, our system can cooperate with existing methods such as S3TC to further compress the texture to a smaller size (Figure 11). That is, our method shrinks homogeneous regions and DXT1 removes redundant codes of each texel block. The two compression methods are independent and can perfectly collaborate together. Figure 11 also shows that DXT1 over compresses the structural region due to its fixed compression ratio. In contrast, our method allows the user to adjust the compression ratio so as to preserve salient features.

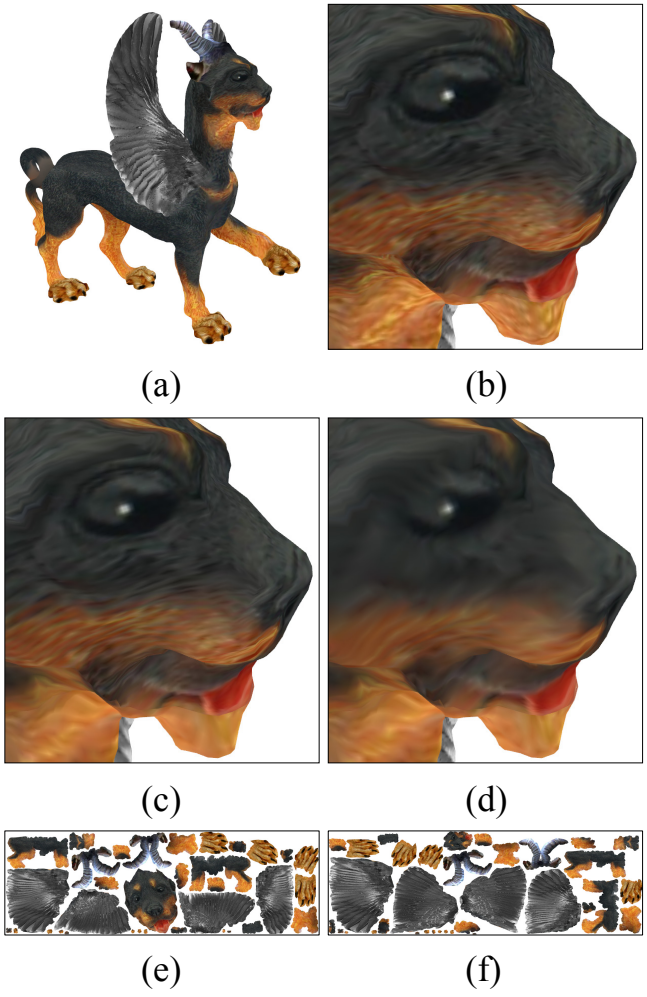

Figure 10. By using our system, users are allowed to control the quality of each texture chart during compression. (a) Feline model. (b)-(d) The feline heads are rendered with the original texture, the compressed texture where texels covered by the head chart are untouched, and the compressed texture that is determined automatically. (e) and (f) show the compressed textures with and without well preservation of head charts.

Figure 11. We render the model with the original texture and the textures compressed by DXT1, our method and DXT1 + our method. In this example, the texture is compressed to have 16.67% and 50% of the original size using DXT1 and our method, respectively. Note that the compression ratio of DXT1 is uncontrollable so that some features are distorted after the compression. We also show that a texture can be compressed by DXT1 and our method simultaneously to achieve a better compression ratio (8.33% of the original size).

A. *Limitations*

Our system squeezes homogeneous regions to reduce texture resolutions, which implies that distortions will increase rapidly if all texels are structural and their gradients are highly disordered. The compression ratio becomes small in this scenario. In addition, we compute each texel color of the compressed texture using trilinear interpolation and suffers from sampling artifacts although the compression ratio is small. Therefore, our system determines texel colors using nearest sampling if the regions are less squeezed to prevent the artifacts.

VII. Conclusions

We have presented a texture compression method based on the importance of each local region. Less structural contents are stored with fewer samples in order to reduce memory and bandwidth consumption. Since the relocation of texture coordinates matches our content aware warp, rendering 3D models using our compressed textures requires no additional decompression cost. Besides, our system compresses each chart individually so as to exactly control the texture quality of different local regions. Semantic features such as human faces also can remain undistorted if necessary. Finally, the compressed charts are packed as tight as possible into a 2D texture to save the space.

Acknowledgment

We thank the anonymous reviewers for their constructive comments. This work was supported in part by the Ministry of Science and Technology, Taiwan (102-2221-E-009 -083 -MY3 and 101-2628-E-009 -020 -MY3).

References

[1] Geetika Tewari, John Snyder, Pedro V. Sander, Steven J. Gortler, and Hugues Hoppe. Signal-specialized parameterization for piecewise linear reconstruction. In *Symposium on Geometry Processing*, volume 71, pages 55–64, 2004.

[2] Graham Campbell, Thomas A. DeFanti, Jeff Frederiksen, Stephen A. Joyce, and Lawrence A. Leske. Two bit/pixel full color encoding. *SIGGRAPH Comput. Graph.*, 20(4):215–223, August 1986.

[3] Günter Knittel, Andreas Schilling, Anders Kugler, and Wolfgang Straser. Hardware for superior texture performance. *Computers and Graphics*, 20(4):475–481, 1996.

[4] Pat Brown. Ext_texture_compression_s3tc. *http://www.opengl.org/registry/specs/EXT/texture_compression_s3tc.txt*.

[5] Pat Brown and Mathias Agopian. Ext_texture_compression _dxt1. *http://www.opengl.org/registry/specs/EXT/texture_compression_dxt1.txt*.

[6] Matt Craighead. Nv_texture_compression_vtc. *http://www.opengl.org/registry/specs/NV/texture_compression_vtc.txt*.

[7] Jacob Strom and Tomas Akenine-Moller. Packman: Texture compression for mobile phones. In *In Sketches program at SIGGRAPH*, 2004.

[8] Jacob Ström and Tomas Akenine-Möller. ipackman: high-quality, low-complexity texture compression for mobile phones. In *Proceedings of the ACM SIGGRAPH/ EUROGRAPHICS conference on Graphics hardware*, HWWS '05, pages 63–70, New York, NY, USA, 2005. ACM.

[9] Jacob Ström and Martin Pettersson. Etc2: texture compression using invalid combinations. In *Proceedings of the 22nd ACM SIGGRAPH/EUROGRAPHICS symposium on Graphics hardware*, GH '07, pages 49–54, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association.

[10] Mark J. Kilgard, Pat Brown, and Yanjun Zhang. Ext_texture_compression_latc. *http://www.opengl.org/registry/specs/EXT/texture_compression_latc.txt*.

[11] Mark J. Kilgard, Pat Brown, and Yanjun Zhang. Ext_texture_compression_rgtc. *http://www.opengl.org/registry/specs/EXT/texture_compression_rgtc.txt*.

[12] ATI Technologies. Ati radeon x800 3dc white paper. *http://www.ati.com/products/radeonx800/3DcWhitePaper.pdf*.

[13] Philipp Klaus Krause. Texture compression. *http://www.colecovision.eu/graphics/texture_compression.pdf*, 2007.

[14] Pedro V. Sander, Steven J. Gortler, John Snyder, and Hugues Hoppe. Signal-specialized parameterization. pages 87–98. The Eurographics Association, 2002.

[15] Duñg T. Võ, Joel Solé, Peng Yin, Cristina Gomila, and Truong Q. Nguyen. Selective data pruning-based compression using high-order edge-directed interpolation. *Trans. Img. Proc.*, 19(2):399–409, February 2010.

[16] Shai Avidan and Ariel Shamir. Seam carving for content-aware image resizing. *ACM Trans. Graph.*, 26(3), 2007.

[17] Ariel Shamir and Olga Sorkine. Visual media retargeting. In *ACM SIGGRAPH Asia Courses*, 2009.

[18] Ran Gal, Olga Sorkine, and Daniel Cohen-Or. Feature-aware texturing. In *Proc. EGSR '06*, pages 297–303, 2006.

[19] Lior Wolf, Moshe Guttmann, and Daniel Cohen-Or. Non-homogeneous content-driven video-retargeting. In *ICCV '07*, 2007.

[20] Yu-Shuen Wang, Chiew-Lan Tai, Olga Sorkine, and Tong-Yee Lee. Optimized scale-and-stretch for image resizing. *ACM Trans. Graph.*, 27(5):118, 2008.

[21] Michael Rubinstein, Ariel Shamir, and Shai Avidan. Improved seam carving for video retargeting. *ACM Trans. Graph.*, 27(3), 2008.

[22] Paul Viola and Michael J. Jones. Robust real-time face detection. *Int. J. Comput. Vision*, 57(2):137–154, May 2004.

[23] Julia A. Bennell and Jose F. Oliveira. The geometry of nesting problems: A tutorial. *European Journal of Operational Research*, 184(2):397 – 415, 2008.

[24] A. Ramesh Babu and N. Ramesh Babu. A generic approach for nesting of 2-d parts in 2-d sheets using genetic and heuristic algorithms. *Computer-Aided Design*, 33(12):879–891, 2001.