透過性を持つ能装束のリアルタイム表現法
Real-time Representing of Noh Gossamer Costume

蔡 康穎, 尹 新, 田中 弘美
Kangying Cai, Xin Yin, Hiromi T.Tanaka
立命館大学
Ritsumeikan University

あらまし:能装束は日本の重要文化財の一つであり, 能装束に関する表現法の研究は活発的に行っている. 本研究は, 透過性を持つ能装束のリアルタイム表現法を提案する. 透過性を持つ能装束を表現するとき, 3 次元の能装束を幾つかレイヤに分割するための計算時間がかかる. ここで, その 3 次元の計算を避け, すでに生成した2次元の情報を用いて計算を行い, 計算時間を減少することができる. また, この手法は能装束以外にも適用できる.

Summary: Noh costume is one important cultural asset in Japan and some studies are carried out to present the appearance of the Noh Costume. In this paper, a real-time representing technique of Noh gossamer costume is proposed. Usually, the 3D Noh costume is divided to some layers to present the gossamer costume and computing time is very long. Here, the 2D information is used to compute new color on the Noh gossamer costume directly and the computing time is decreased. This technique can be used to represent other transparent objects also.

キーワード:能装束, リアルタイム, 表現, コンピュータグラフィックス
Keywords: Noh costume, real-time, representing, computer graphics

1. Introduction

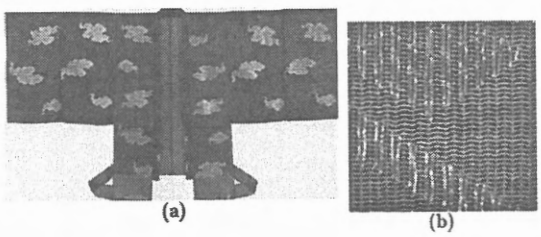

Figure 1: *(a) Example of Japanese Noh Gossamer Costume. (b) Closeup of gossamer textile.*

Noh is a classical Japanese performance form which combines elements of dance, drama, music and poetry into one highly aesthetic stage art. It has been performed since the 14th century and is still popular throughout Japan now. Currently there exists many ancient Noh costumes. Some photos of Noh costumes are shown in Figure 1. The oldest surviving Noh garments date from the 15th century during the reign of the sixth shogun, Yoshimasu (reigned 1440-1473). For protection reason, these precious culture heritage can not be exhibited frequently. The aim of our research is to digitalize these heritages and provide an interactive and flexible observation way to the users. We think it will be helpful to make more and more people become familiar with Japanese Noh art. There are two problems need to be addressed, how to represent the surface appearance of the woven fabric and how to provide interactive walkingthrough.

Fabric's mesostructure plays an extremely important role in the reflectance behavior of cloth. According to how the mesostructure is reconstructed, cloth rendering techniques can be divided into two types. The first strategy explicitly

models the mesostructure, renders it using different lighting models and rendering techniques and thus lack generality. The second category measures the reflectance properties of a given textile and then uses the result for realistic rendering. The optical behavior of the yarns and the fabric mesostructure can be described by the bidirectional texture function (BTF), which describes how a planar texture probe changes its appearance under different illuminating and viewing directions. Considering of the uncertainty of yarn material and weave of Noh costume, we follow the second technique which is based on the measured BTF data.

In this paper, we focus on processing the Noh costumes made by gossamer textile which refers to those with loose open weave, as shown in Figure 1. The significant intervals between yarns make the gossamer Noh costume highly transparent. We achieve the interactive transparency rendering of Noh gossamer costume by depth peeling technique [9], which is a simple and robust algorithm for resolving order-independent transparency rendering on commodity graphics hardware. Depth peeling approach does not require sorting geometric primitives and can deal with intersecting polygons. However, its principal drawback is that a scene with maximum depth complexity D must be rendered D times. We enhance our depth peeling based transparency rendering in two effective ways to allow fast walkingthough.

Most of current depth peeling based algorithms perform the z-range culling step in fragment shader. Thus the whole model need to be geometric transformed and rasterized in each rendering pass.

Making use of the horsepower of geometry shader, the latest programming component of GPU, we discard the polygons which have been wholly selected and rendered in geometry shader to avoid the unnecessary rasterization of them. Thus each rendering pass except the first one can be accelerated.

Our system further accelerate the interactive rendering of Noh gossamer costume by taking advantage of the coherence between adjacent frames.

In most cases, the appearance of the scene changes little from frame to frame when a viewer navigates through the virtual environment. We exploit this path coherence by caching one frame for possible reuse in many subsequent frames. The depth images representing different layers of the source frame are warped to current frame and then resorted to get correct transparency under current viewpoint.

Through this way, the complexity of our interactive rendering algorithm only relates with the resolution of the output window. Considering of the big number of polygons needed to describe various details of cloth (drape, crease etc.) and the possible several transparent objects in the scene, our interactive rendering efficiency can be improved through making use of the path coherence. The interactivity of our visualization system is also guaranteed by executing the depth image warping operation on GPU. No retrieval from GPU to CPU is necessary.

An error metric that quantifies the discrepancy between the appearance of the cached depth images and original geometry is used to determine whether the warped depth images are likely to provide an adequate approximation of current frame. The error is calculated for each fragment and also accomplished by GPU. We use occlusion query [7] to know the number of fragments whose error for current frame is larger than some user specified threshold. When there are too many 'fault' fragments (also indicated by some user defined number), the next frame is rendered by the original geometry and exactly BTF mapping and cached as the new source frame.

The difference between our accelerated interactive visualization technique and other depth image based rendering (DIBR) algorithm such as LDI [21] is that our approach does not require a pre-sampling stage while it is necessary for most other DIBR systems. Most DIBR systems do not use scene geometry during interactive rendering no matter whether it is available or not. However, our system reuse the scene geometry when the cached data cannot meet the requirement of rendering current frame.

The main contribution of our approach is as follows:

- Propose an system of digitalization and interactive visualization of the culture heritage, Japanese ancient Noh gossamer costumes.
- Successful accelerate depth peeling algorithm for order-independent transparency rendering by getting rid of the rasterization of those polygons have already been wholly selected in the previous rendering passes. To our knowledge, other depth peeling approaches haven't included this effective step.
- Gain an satisfied speedup of walkingthrough by successful combination of geometry-based and image-based rendering. The switch between the two rendering strategies is flexibly controlled by the per frame error calculated on GPU. No pre-computation is needed.

2. Related work

2.1. Visualization of woven cloth

Modeling and rendering of textiles and clothing keep being an active research area in computer graphics because of their widespread presence in our daily life. Generally speaking, cloth can be divided into two types, knitwear and woven cloth. Rendering of knitwear focus on capture the fluffy nature of wool and the resulting self-shadowing due to the volume occupied by thick woolen yarn [10,8,6]. For woven cloth, the complex weave patterns needs special representation and creation techniques. [2, 3] addresses the problem of visualizing arbitrary complex weave patterns. [22] achieve the efficient and realistic visualization of cloth by using BTF.

Instead of fitting some BRDF model, [22] interactively renders cloth by using a principal component analysis of the captured BTF data. Our system adopts the same strategy for reproducing surface appearance of Noh costume. We also resolve the transparency rendering of cloth, which hasn't been addressed by [22].

2.2. Transparency rendering

The major challenge for rendering transparency is that, to achieve the correct compositing effect, the entire model needs to be resorted whenever the view point or scene geometry changes. On the contrary, only the front-most parts influence the rendering of opaque objects, which is a minimum-finding problem. Sorting is an time consuming and tedious step especially for complex scene and constantly deforming objects.

There exists a variety of techniques that exploits the massive computation power of the z-buffer graphics hardware to implement the crucial sorting task for transparency rendering.

Depth peeling [16,9] is the most widely used one due to its simplicity and robustness. The technique requires $O(N)$ rendering passes to render order-independent transparency with depth complexity N. The total time complexity is $O(N^2)$ as each render pass requires passing down all transparent geometry. [23] improves the time complexity of depth peeling from quadratic to linear, but requires the application to sort objects into depth batches. [15]proposes an acceleration by peeling multiple layers simultaneously per rendering pass via multiple rendering target (MRT) technique and read-modify-write (RMW) operations, which is similar with [4]'s k-Buffer. Other improvements, such as [24,1], lack generality as they require features not available in current generation commodity graphics hardware. Exploiting the power of latest geometry shader, we propose another effective enhancement of depth peeling technique without any hardware modification requirement. Our improvement of depth peeling can be easily integrated into other depth peeling based method.

2.3. Coherence in computer graphics

Coherence is based on the principle of locality, whereby "nearby" things do have the same or similar characteristics. Coherence properties have been exploited in a variety of different methods and techniques [10,12,21,17,5]. [10] exploits path coherence to accelerate walkthroughs of geometrically complex scenes. In the course of a walkthrough, images recently used are cached for reuse in subsequent frames. The visual artifacts are controlled by an error metric. The basic idea behind our acceleration method for walkingthough is similar with [21]. However, instead of estimating the error in object space as in [21], our system qualifies the discrepancy by directly evaluating the rendered image by using the horsepower of GPU.

3. Measurement

We carried out the measurements using Optical Gyro Measuring Machine (OGM). As shown in Figure 2, OGM includes a four-axis motion device that can position a camera and a light source independently anywhere on the hemisphere under computer control. The measuring and post-processing step of the BTF data of gossamer textile is similar with [22].Please refer to [22] for the details. The only difference is that the textile needs to be separated from the background in all the captured images. By using black background, most of the separation work can be automatically accomplished by comparing each pixel's RGB value with some threshold.

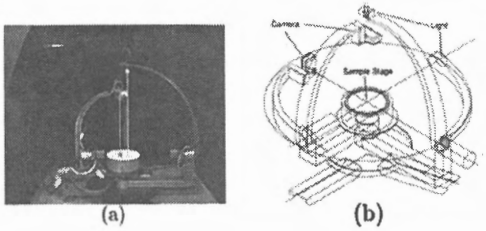

Figure 2: *(a) OGM. (b) Illustration of OGM's motion device.*

4. Transparency rendering

As mentioned earlier, our solution to order independent transparency rendering of gossamer Noh costume is based on the depth peeling technique. For clarity, we first give a brief introduction of the classical depth peeling approach and then describe our effective improvements using geometry shader.

4.1. Brief Review of Depth Peeling

Depth peeling, described in general by Mammen [1] and for GPUs by Everitt [9], solves order independent transparency on the GPU by multi-pass rendering. Within i^{th} pass the i^{th} layer (nearest to the eye) is selected and rendered. The layers-so-far buffer is maintained to accumulate the transparent layers already rendered. Besides the ordinary z-buffering, each pass performs an additional z-range culling, which only accepts fragments that are behind the corresponding z in the layers-so-far buffer, in fragment shader. The peeling process stops until no more transparent fragments are rendered (this can be determined by occlusion query [7]).

The principle drawback of depth peeling is that the algorithm needs $O(N)$ rendering passes where N is the layer

number of transparent fragments in the scene. Each pass requires a complete geometric transformation and rasterization of all transparent polygons. Some scenes, such as particle systems for smoke, require $O(N^2)$ rendering time, where N is the number of primitives and can be very huge.

4.2. Z-range culling in geometry shader

Most of the existing depth peeling methods perform the z-range culling operation and discard fragments in fragment shader. Then those geometric primitives have already been wholly selected and rendered also need to be transformed and rasterized in the remaining rendering passes, which is meaningless and should be avoided. However, before the geometry shader is available, it is almost impossible to do z-range culling operation in vertex shader as one polygon can fall into several layers and usually each vertex shader can not fetch the information of the corresponding vertex.

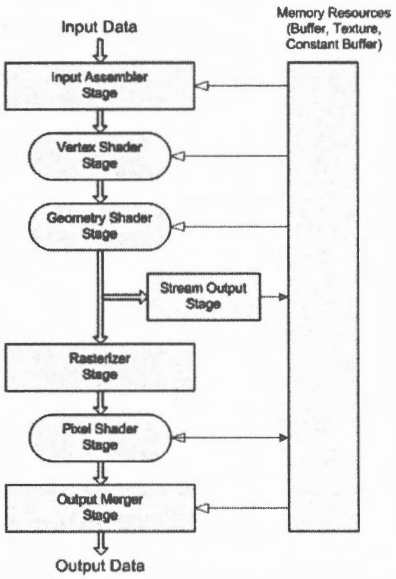

Figure 3: *Picture from DirectX 10 documentation that represents the graphic pipeline.*

Nvidia opened geometry shader at November, 2006. Geometry shader stage take place between vertex shader and viewport clipping stage, as shown in Figure 3. It operates on the assembled primitives transformed by vertex shader and can emit zero or more primitives, which are rasterized and ultimately passed to pixel shader. Thus we can perform another z-range culling in geometry shader and forbid those polygons whose vertices are all before the corresponding z in the layers-so-far buffer to be passed to the viewport clipping stage. Those unnecessary rasterization can be avoided in this way. Our effective enhancement does not require any hardware modifications and can be integrated into other acceleration techniques of depth peeling such as [23,15].

5. Interactive visualization

As a viewer follows a continuous path through a virtual environment, there is typically considerable coherence between successive frames. Thus we are activated to exploit this coherence to accelerate the interactive rendering of gossamer Noh costume. Our system caches the depth images representing various layers rendered in one frame and uses them to render several subsequent frames. Instead of simply reusing the same images, we warp the cached depth images to current view point.

5.1. Depth image warping on GPU

Depth image warping, one of the key techniques of Image-Based rendering (IBR), transforms the pixels of the source image to the target image by using some 3D warping equation. The most widely used 3D warping equation is developed by McMillan and Bishop [18,19], McMillan and Bishop emphasized a non-Euclidean formulation of 3D warping, which is useful for warping images acquired with unknown or poorly-known camera calibration [19]And their equation doesn't compute the target pixel depth explicitly. However, our interactive rendering system can easily fetch the "camera" parameters and needs the warped depth to get the correct transparency of the target frame.

As shown in Figure 5, for a given target viewpoint, the 3D warping can be divided into two steps, first reverse project the source image to 3D object space and then re-project them to the target window space. OpenGL/Direct3D, the premier environments for developing interactive graphics applications, perform the similar procedure when rendering 3D objects. Thus, same as [14], we adopt another 3D warping equation which is accordant with the rendering pipeline of OpenGL/Direct3D. The vertex transformation in rendering pipeline is shown in Figure 5.

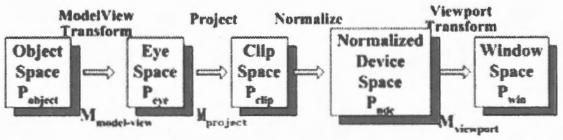

Figure 4: *The vertex transformation in rendering pipeline.*

Suppose $\mathbf{P}_{object} = [x_o, y_o, z_o, 1]^T$, we have

$$\mathbf{P}_{eye} = [x_e, y_e, z_e, 1]^T = \mathbf{M}_{model-view} \times \mathbf{P}_{object} \quad (1)$$

$$\mathbf{P}_{clip} = [x_c, y_c, z_c, w_c]^T = \mathbf{M}_{project} \times \mathbf{P}_{eye} \quad (2)$$

$$\mathbf{P}_{ndc} = [x_{ndc}, y_{ndc}, z_{ndc}, 1]^T = \mathbf{P}_{clip}/w_c \quad (3)$$

$(-1 \leq x_{ndc}, y_{ndc}, z_{ndc} \leq 1)$

$$\mathbf{P}_{win} = [x_w, y_w, z_w, 1]^T = \mathbf{M}_{viewport} \times \mathbf{P}_{ndc} \quad (4)$$

Suppose the window width and height are W and H pixels respectively, $0 \leq x_{win} \leq W$, $0 \leq y_{win} \leq H$ and $0 \leq z_{win} \leq 1$. Smaller z means nearer to the viewpoint.

Combine equation $1 \rightarrow 4$, we can get the transform equation from object space to window space

$$\mathbf{P}_{win} = \mathbf{M}_A \times \mathbf{P}_{object} \quad (5)$$

$$\mathbf{M}_A = 1/w_c \times \mathbf{M}_{viewport} \times \mathbf{M}_{projection} \times \mathbf{M}_{model-view} \quad (6)$$

The 3D warping equation can be written as

$$\mathbf{P}_{dwin} = \mathbf{M}_{warping} \mathbf{P}_{swin} \quad (7)$$

$$\mathbf{M}_{warping} = \mathbf{M}_{tA} \mathbf{M}_{sA}^{-1} \quad (8)$$

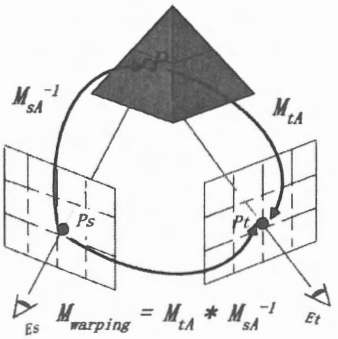

Figure 5: *The 3D warping equation accordant with the rendering pipeline.*

By using equation 8, each pixel can perform its 3D warping independently and efficiently on GPU although the computation seems complex. And all the necessary parameters can be easily fetched through some 3D API. We use a vertex shader to perform this step as a vertex shader enables user to freely define the transformation matrix of vertices. After all the depth images are warped, a fragment shader resort and then blend the resultant depth images. Through this way the complexity for getting correct transparency rendering changes from N^2 to n^2, where N is the number of polygons of the scene geometry and n is the window resolution.

5.2. Error Metric

As described above, our system warps every pixel in a geometrically correct manner as we use images with per-pixel depth (depth images). However, the mapping between source and target depth image pixels is not one to one.

- Folds : As shown in Figure 6 (a). P_t and Q_t should be blended according to their depth. However, there is writing crash problem when we do 3D warping by vertex shader, the correct blending can not be guaranteed in this case.
- Holes : As shown in Figure 6 (b) and (c). Holes can be filled by rasterization. However, it will decrease the image quality.

Thus we need an error metric, which, given source and target view point and source depth images, quantifies the difference between the appearance by warping depth images and rendering the actual geometry. If the difference is smaller than some user-specified threshold ε, the approximation is deemed acceptable. An important requirement for an acceptable error metric is that it must be suitable for parallel computation as we want to perform all the computation on GPU.

Our error metric is

$$\mathbf{Err} = \sum_{i=1}^{i=D} |N_{ti} - N_{si}| + \max(\left|\overrightarrow{P_{ti}Q_{ti}}\right| - \left|\overrightarrow{P_{si}Q_{si}}\right|) \quad (9)$$

$$+ \max(\left|\overrightarrow{P_{ti}E_t}\right| - \left|\overrightarrow{P_{si}E_s}\right|) + \max(\left|\overrightarrow{P_{ti}P_{si}}\right|)$$

where D is the number of layers, s means the source frame and t means the target frame.

- $\sum_{i=1}^{i=D} |N_{ti} - N_{si}|$. N_{si} and N_{ti} are the numbers of non-empty pixels in depth image i, i.e. layer i, of source and target frame respectively.
  This item is used to qualify folds. N_{si} and N_{ti} are fetched by occlusion query, which is also the necessary step of classical depth peeling.
- $\max(\left|\overrightarrow{P_{ti}Q_{ti}}\right| - \left|\overrightarrow{P_{si}Q_{si}}\right|)$. P_{si} and Q_{si} are any pair of adjacent non-empty fragments in the same depth image i of source frame. P_{ti} and Q_{ti} are in target frame.
  This item measures the distance of two adjacent fragments after 3D warping. We use it to quality the holes. $\left|\overrightarrow{P_{si}Q_{si}}\right|$ always equals to 1 (pixel). $\left|\overrightarrow{P_{ti}Q_{ti}}\right|$ needs to be evaluated in geometry shader as it needs the information of at least two pixels. We use a geometry shader in the rendering pass which performs depth image warping to calculate this item.
- $\max(\left|\overrightarrow{P_{ti}E_t}\right| - \left|\overrightarrow{P_{si}E_s}\right|)$. E_s and E_t are the source and target view points respectively.
  This item measures change of the distance between the depth image pixel and the viewpoint. We adopt this item

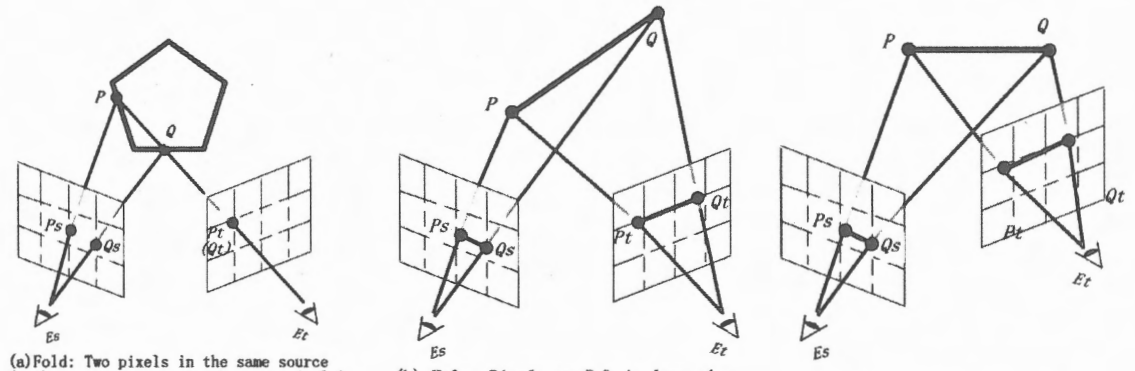

Figure 6: *The mapping between source and target depth image pixels is not one to one.*

to estimate whether there are some parts not fallen into the view frustum of the source view point visible to the target view point. $|\overrightarrow{P_{si}E_s}|$ is calculated by a fragment shader during rendering the source frame and stored into a texture. $|\overrightarrow{P_{ti}E_t}|$ and $|\overrightarrow{P_{ti}E_t}| - |\overrightarrow{P_{si}E_s}|$ are calculated in the vertex shader performing 3D warping.

- $\max(|\overrightarrow{P_{ti}P_{si}}|)$.
  This item is designed to qualify the error of BTF texture mapping. When the window space coordinates of a vertex doesn't change much in source and current frame, we avoid texture mapping operation for current frame. This item can also be calculated in the vertex shader performing 3D warping.

As it is hard to perform on-the-fly operations such as sum or maximum on GPU, we translate max operation in equation 9 to comparison operation, i.e. comparing the corresponding value to some user-defined threshold for each item, and discard the primitives fail to pass the test. Then the result of equation 9 can be returned to our system through occlusion query.

When the warping result of cached depth images cannot guarantee the rendering quality of current frame, we use the scene geometry and BTF data to render the current frame. Most DIBR system prepares source depth images in precomputation stage and do not use scene geometry during interactive walkingthrough. We choose to re-render the scene using the original geometry when necessary because the precomputation stage can be omitted and new objects can be added to our scene during interactive walkthrough.

6. Experimental Result

We have done some experiments to evaluate our approach. The results were obtained under Windows XP Service Pack 2 on a 2.80GHz Pentimue, 2GB RAM machine. Our graphics accelerator is Nvidia GeForce 8600 GT with 256MB onboard memory. The Noh costume model we used is scanned by VIVID 910 and the Noh costume belongs to the Tsubouchi Memorial THEATRE MUSEUM, Waseda University, Japan. The model includes 76,212 vertices, 150,013 triangles and 4 layers in most cases. Figure 7 shows the visual result of our system, rendered at a 512×512 resolution. Comparing with the traditional interactive visualization method, i.e. update each frame with depth peeling and re-generating view-dependent texture mapping our accelerated method can gain a acceleration of 40%. The frame rate can be improved from around 8 FPS to almost 12 FPS. We set the threshold of $(|\overrightarrow{P_{ti}Q_{ti}}| - |\overrightarrow{P_{si}Q_{si}}|)$, $(|\overrightarrow{P_{ti}E_t}| - |\overrightarrow{P_{si}E_s}|)$, and $(|\overrightarrow{P_{ti}P_{si}}|)$ as 4 pixels, $0.2 \bullet |\overrightarrow{P_{si}E_s}|$, and $(0.1 \bullet |\overrightarrow{P_{ti}P_{si}}|)$ respectively. The primitives which can not pass any of the last three items of comparison of Equation 9 is discarded. Then the result of 9 can be evaluated by $\sum_{i=1}^{i=D} |N_{ti} - N_{si}|$ where N_{ti} is known by occlusion query. Our system will render the next frame by traditional method if $\sum_{i=1}^{i=D} |N_{ti} - N_{si}| < 0.1 \sum_{i=1}^{i=D} N_{si}$

7. Conclusion

We have presented a system for interactive visualizing the precious ancient Japanese Noh gossamer costume. Our system use the bidirectional texture function (BTF) acquired by OGM and depth peeling technique to achieve the realistic transparency rendering of Japanese Noh gossamer costume. We also adopt two effective acceleration method in our system to guarantee the interactive walkthrough.

- Do the z-range culling operation in both geometry and fragment shader. The un-necessary rasterization of those already wholly peeled polygons can be avoided.
- Making use of the coherence between adjacent frames, cache the result of one frame and reconstruct the next several frames using the cached data. Suppose scene geometry has N polygons and the output window has n resolution. Comparing with the traditional rendering method,

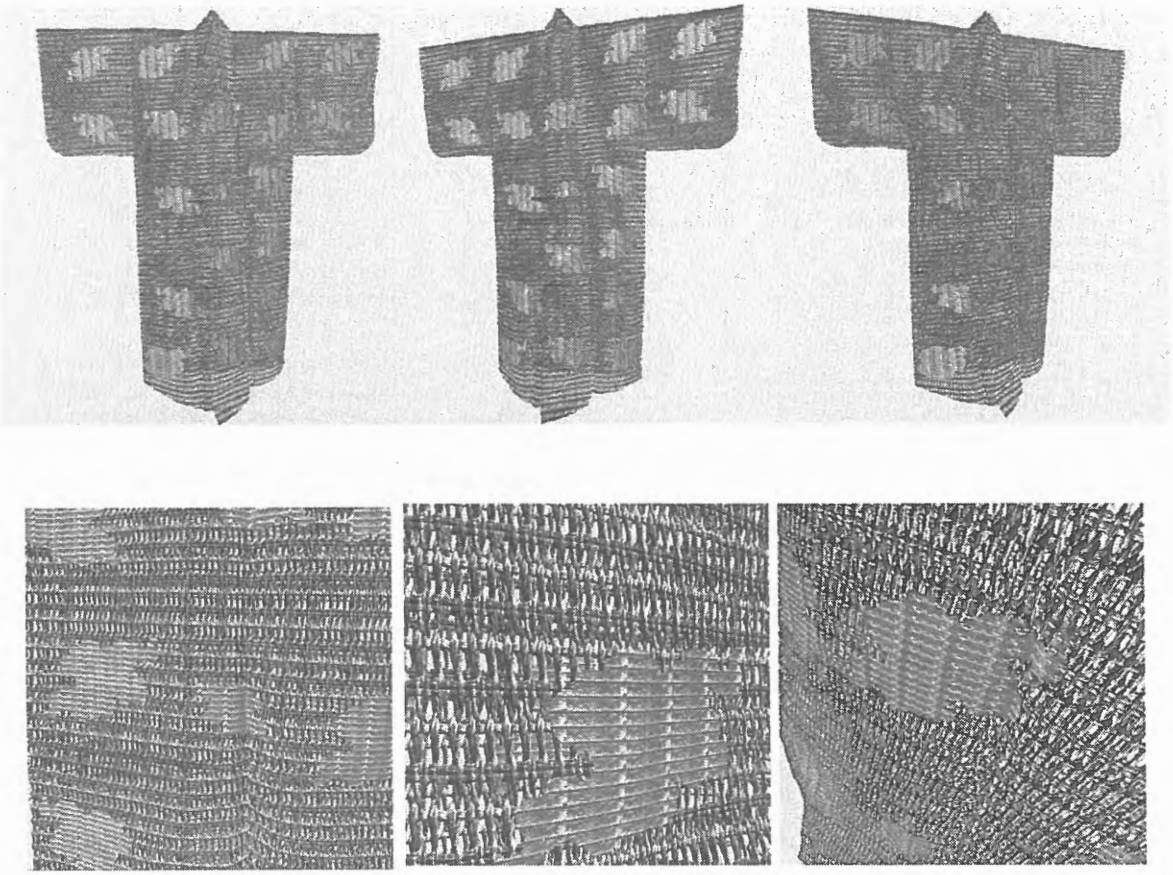

Figure 7: *Interactive rendering result of our system.*

the time complexity of our interactive rendering scheme changes from N^2 to n^2 and has no relationship with the input complexity. Note that $N > n$ in most cases.

We only consider the transparency of gossamer textile in this paper. In the future we will concentrate on how to achieve efficient translucency rendering.

Acknowledgments

This work was supported partly by the Grants-in-Aid for Scientific "Research Scientific Research(A) 17200013" and "Encouragement of Young Scientists(B) 19700104" of Japan Society for the Promotion of Science. This work was also supported partly by "Kyoto Art Entertainment Innovation Research" of Centre of Excellence Program for the 21st Century of Japan Society for the Promotion of Science and "Research for Promoting Technological Seeds 09-14-001" of Japan Science and Technology Agency.

References

[1] Adabala, N., Magnenat-Thalmann, N. and Fei, G.: Real-time rendering of woven clothes, *VRST '03: Proceedings of the ACM symposium on Virtual reality software and technology*, New York, NY, USA, ACM Press, pp. 41–47 (2003).

[2] Adabala, N., Magnenat-Thalmann, N. and Fei, G.: Visualization of woven cloth, *EGRW '03: Proceedings of the 14th Eurographics workshop on Rendering*, Aire-la-Ville, Switzerland, Switzerland, Eurographics Association, pp. 178–185 (2003).

[3] Aila, T., Miettinen, V. and Nordlund, P.: Delay streams for graphics hardware, *ACM Trans. Graph.*, Vol. 22, No. 3, pp. 792–800 (2003).

[4] Bavoil, L., Callahan, S. P., Lefohn, A., Jo a. L. D. C. and Silva, C. T.: Multi-fragment effects on the GPU using the k-buffer, *I3D '07: Proceedings of the 2007 symposium on Interactive 3D graphics and games*, New York, NY, USA, ACM Press, pp. 97–104 (2007).

[5] Bittner, J., Wimmer, M., Piringer, H. and Purgathofer, W.: Coherent Hierarchical Culling: Hardware Occlusion Queries Made Useful, *Computer Graphics Forum*, Vol. 23, No. 3, pp. 615–624 (2004).

[6] Chen, Y., Lin, S., Zhong, H., Xu, Y.-Q., Guo, B. and Shum, H.-Y.: Realistic Rendering and Animation of Knitwear, *IEEE Transactions on Visualization and Computer Graphics*, Vol. 09, No. 1, pp. 43–55 (2003).

[7] Craighead, M.: Nv occlusion query, *http://oss.sgi.com/projects/ogl-sample/registry/nv/occlusion_query.txt* (2002).

[8] Daubert,K., Lensch, H. P. A., Heidrich, W. and Seidel, H.-P.: Efficient Cloth Modeling and Rendering, *Proceedings of the 12th Eurographics Workshop on Rendering Techniques*, London, UK, Springer-Verlag, pp. 63–70 (2001).

[9] Everitt, C.: Interactive order-independent transparency, *Technical report, NVIDIA Corporation, http://www.nvidia.com/* (2001).

[10] Gröller,E., Rau, R. T. and Straβer, W.: Modeling and Visualization of Knitwear, *IEEE Transactions on Visualization and Computer Graphics*, Vol. 1, No. 4, pp. 302–310 (1995).

[11] Gröller, M. E.: Coherence in Computer Graphics, PhD Thesis, Institute of Computer Graphics and Algorithms, Vienna University of Technology, Favoritenstrasse 9-11/186, A-1040 Vienna, Austria (1992).

[12] Hubschman, H. and Zucker, S. W.: Frame-to-frame coherence and the hidden surface computation: Constraints for a convex world, *SIGGRAPH Comput. Graph.*, Vol. 15, No. 3, pp. 45–54 (1981).

[13] Im, Y.-H. and Han, C.-Y.: A Method to Generate Soft Shadows Using a Layered Depth Image and Warping, *IEEE Transactions on Visualization and Computer Graphics*, Vol. 11, No. 3, pp. 265–272 (2005). Member-Lee-Sup Kim.

[14] Jiang, Z., Bao, H. and tsin Wong, T.: Rendering Driven Depth Image Quantization and Compression, *Technical Report TR-2004-01, State Key Lab. of CAD and CG, Zhejiang University, China* (2004).

[15] Liu, B., Wei, L.-Y. and Xu, Y.-Q.: Multi-Layer Depth Peeling via Fragment Sort, *MSR-TR-2006-81* (2006).

[16] Mammen, A.: Transparency and Antialiasing Algorithms Implemented with the Virtual Pixel Maps Technique, *IEEE Comput. Graph. Appl.*, Vol. 9, No. 4, pp. 43–55 (1989).

[17] Mark, W.: Post-Rendering 3D Image Warping: Visibility, Reconstruction, and Performance for Depth-Image Warping, Technical report, Chapel Hill, NC, USA (1999).

[18] McMillan, L. and Bishop, G.: Head-tracked stereoscopic display using image warping, *Proc. SPIE 2409A* (1995).

[19] McMillan, L.: An Image-Based Approach to Three-Dimensional Computer Graphics, Technical report, Chapel Hill, NC, USA (1997).

[20] Sattler, M., Sarlette, R. and Klein, R.: Efficient and realistic visualization of cloth, *EGRW '03: Proceedings of the 14th Eurographics workshop on Rendering*, Aire-la-Ville, Switzerland, Switzerland, Eurographics Association, pp. 167–177 (2003).

[21] Shade, J., Gortler, S., wei He, L. and Szeliski, R.: Layered depth images, *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, New York, NY, USA, ACM Press, pp. 231–242 (1998).

[22] Shade, J., Lischinski, D., Salesin, D. H., DeRose, T. and Snyder, J.: Hierarchical image caching for accelerated walkthroughs of complex environments, *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, New York, NY, USA, ACM Press, pp. 75–82 (1996).

[23] Wexler, D., Gritz, L., Enderton, E. and Rice, J.: GPU-accelerated high-quality hidden surface removal, *HWWS '05: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, New York, NY, USA, ACM Press, pp. 7–14 (2005).

[24] Wittenbrink, C. M.: R-buffer: a pointerless A-buffer hardware architecture, *HWWS '01: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*, New York, NY, USA, ACM Press, pp. 73–80 (2001).