A Perceptually-Based Texture Caching Algorithm for Hardware-Based Rendering

Reynald Dumont Fabio Pellacini James A. Ferwerda
Program of Computer Graphics, Cornell University

Abstract: The performance of hardware-based interactive rendering systems is often constrained by polygon fill rates and texture map capacity, rather than polygon count alone. We present a new software texture caching algorithm that optimizes the use of texture memory in current graphics hardware by dynamically allocating more memory to the textures that have the greatest visual importance in the scene. The algorithm employs a resource allocation scheme that decides which resolution to use for each texture in board memory. The allocation scheme estimates the visual importance of textures using a perceptually-based metric that takes into account view point and vertex illumination as well as texture contrast and frequency content. This approach provides high frame rates while maximizing image quality.

1. Introduction

Many important graphics applications require complex scenes to be rendered at interactive rates (simulation, training systems, virtual environments, scientific visualization, games). Hardware-based rendering is currently the best solution for these interactive applications. Performance increases in hardware-based graphics accelerators have enabled significant improvements in rendering capabilities, but concurrent increases in user requirements for realism, complexity and interactivity mean that computational demands will continue to outstrip computational resources for the foreseeable future. For example, the performance of current graphics hardware strongly depends on the number of primitives drawn as well as the number and resolution of the textures used to enrich the visual complexity of the scene. While much work has been done to try to reduce the number of primitives displayed (see [12] for a good summary), little research has been devoted to optimizing texture usage.

Current graphics accelerators employ fast memory for texture storage. To achieve the best possible framerate, all the textures should reside in texture memory. While textures might be dynamically loaded from main memory, this remains a slow operation that causes drastic framerate reductions (even with fast AGP buses). Hardware developers are trying to address this problem by constantly incrementing the amount of texture memory available, by speeding up texture swapping operations and by employing hardware texture compression techniques. However such improvements do not solve the problem when the total size of textures exceeds the capacity of the board memory. In such conditions, it is often impossible to allocate on-board memory quickly enough to load the textures needed to render the scene.

A few software texture caching systems have been presented in the past to address this problem. Some of them optimize texture swapping with respect to no image degradation. While these algorithms ensure image quality, they provide framerates which are strongly dependent on the size of the original texture set. Other approaches guarantee target framerates by allowing image degradation. Unfortunately, the metrics employed to measure image degradation are too simple and do not guarantee that the rendered image has the best possible quality for the given target framerate.

delivered by
EUROGRAPHICS DIGITAL LIBRARY
www.eg.org diglib.eg.org

In this paper, we present a new algorithm for texture caching that allows fast and predictable framerates while maximizing image quality on current low-end graphics hardware. The algorithm employs a resource allocation scheme that decides which resolution to use for each texture in board memory. The resolution is chosen depending on the current view-point and illumination conditions as well as texture contrast and frequency content. This naturally led us to employ a perceptual metric. Unlike previous approaches, the texture content is analyzed to provide the best decisions on the chosen resolutions. Depending on the texture content, the allocation scheme allows more or less reduction in resolution for the texture to save on-board memory.

In the following sections, we first review previous work and then outline our texture caching algorithm before describing some of the implementation details. Finally, we present the results produced by the algorithm, before concluding and discussing future work.

2. Related work

Hardware texture compression is now frequently used to increase the effective size of texture memory in graphics hardware. A simple lossy scheme presented by S3 [18] can now be found in most off-the-shelf graphics boards. Talisman [19] is an example of non-standard graphics pipeline that employs a hardware-based compression scheme similar to JPEG. A texture compression algorithm based on vector quantization has been proposed to be used in hardware in [1].

Software caching schemes try to address texture memory limitations by only using a subset of the original texture set to render the current frame. A good portion of the texture caching algorithms described in the literature uses specialized caching schemes to address specific applications. For examples, Quicktime VR [4] cuts texture panoramas into vertical strips for caching purposes. Many simple metrics, based on viewing distance and viewing angle, have been proposed in terrain visualization applications [2][6][10][15]. A progressive loading approach has been presented in [5] and applied to terrain visualization; this caching scheme tolerates image degradation to ensure framerate during its progressive loading steps.

While these approaches have proven to be fairly effective, either they do not guarantee framerate, or when they do, they cannot guarantee that the image generated is the best possible one since their metrics do not take perceptual effects into account. In this paper we will show that a caching stategy that maximizes image quality by using a perceptual metric is better than standard load/unload priority schemes.

3. Texture cache formulation

3.1. Problem statement

We consider that the color at each pixel of each rasterized polygon is the product of the Gouraud interpolated vertex color multiplied by the color of the trilinearly interpolated texture applied to the polygon (using a pyramidal mip-mapping scheme for each texture). We can write the shading equation for each pixel (x,y) as:

$$C^{x,y} = V^{x,y} \cdot T^{x,y} \qquad (1)$$

where C is the pixel color, V the Gouraud interpolated vertex color and T is the trilinearly interpolated texture color.

In order to maximize the framerate, we have to ensure that the texture set in use is smaller than the texture memory on the graphics board. When this is not possible, the

scene should be displayed with a texture set that maximizes perceived image quality, while respecting texture memory constraints. To obtain this set, we can use the mip-map pyramids and only load a subpart of each original pyramid (called from now on subpyramid) on the board.

In order to solve this resource allocation problem, we developed an algorithm based on a cost/benefit analysis, following the formalism presented in [8]. We define a texture tuple as (T_i, j_i) to be the instance of a texture mip-map pyramid T_i rendered using a subpyramid starting at level j_i (higher values of j_i correspond to lower resolution). For each texture subpyramid we also define a cost function c_i and a benefit function q_i. The cost of a subpyramid is its size, while its benefit is computed by our perceptually-based error metric which predicts the expected visual degradation between the image rendered by the texture subpyramid (T_i, j_i) and the one for the high-resolution "gold standard" texture pyramid (T_i, v_i) ($0 \le v_i < j_i$). Our resource allocation scheme maximizes the total benefit Q, while keeping the total cost C smaller than texture memory limits. Using this formalism we can state our resource allocation problem as

Maximize: $$Q = \sum_i q_i \quad (2)$$

Subject to: $$C = \sum_i c_i \le texture_memory \quad (3)$$

3.2. Resource allocation algorithm

In general the problem of maximizing the total benefit under the cost constraint is NP-complete, so it cannot be solved in real time even for a small number of textures. While approximation algorithm exists [17], they are computationally inadequate for a real-time implementation. After trying various approaches, we settled on an improved greedy algorithm.

We first sort all the possible texture subpyramids with respect to $\Delta q_i(j_i, j_{i+1}) / \Delta c_i(j_i, j_{i+1})$ (estimated degradation between level j_i and j_{i+1} divided by memory saved while using level j_{i+1} instead of level j_i). Starting with the full texture set, we keep reducing its size by discarding the texture levels that have smaller $\Delta q_i(j_i, j_{i+1}) / \Delta c_i(j_i, j_{i+1})$ until the size of the set is smaller than the allowed size (i.e board texture memory).

3.3. Perceptually-based benefit function

The major contribution of our work is the metric used to estimate the benefit function. Given the current viewpoint, we would like this metric to accurately measure the perceived visual difference between the current subpyramid and the texture pyramid used in the gold standard. We believe that a metric based on a psychophysical model of the human visual system is best suited to accurately measure visual differences in images. In the past this approach has been employed very successfully to speed up offline algorithms [3][13][14][16]. Unfortunately the computational cost of such metrics prohibits their use for real time applications.

However, our application does not require the same level of generality in perceptual metrics as the ones presented before, since we are specifically interested in computing the visible difference caused by decreasing the resolution of a texture. This means that we can derive a computationally efficient metric by tailoring the metric to the needs of our application without losing the predictability and accuracy of the perceptually-based metrics previously given in the literature.

Our benefit function is based on two aspects of the human visual system: *visual saliency* (α_i) and *perceived error* (ε_i). Intuitively, the benefit q_i of a rendered texture is proportional to the probability α_i that we are focusing our attention to this specific texture, and the visual degradation ε_i that may occur by reducing its resolution.
We can formally write this as:

$$q_i = \alpha_i \cdot (-\varepsilon_i) \quad (4)$$

In our formulation, the maximum benefit is 0 when we are using the gold standard texture pyramid and it decreases when we use lower resolution subpyramids.
Following [10], we model visual saliency as proportional to the pixel coverage of the texture in the current frame. This is a statistical model based on the premise that we are focusing our attention on each part of the image with equal probability.
We can simply write

$$\alpha_i \propto A_i \quad (5)$$

where A_i is the area in pixel covered by texture T_i.
More advanced saliency models might avoid cases where small areas present strongly perceivable errors. Metrics for visual saliency are a new area of research in computer graphics. [20] has introduced a sophisticated metric based on low-level visual processing and a model of attention. However this metric is currently too computationally expensive to be used in interactive systems.
The perceived error term ε_i measures the apparent visual difference between the image rendered using the gold standard texture pyramid and the one rendered using a subpyramid. In order to estimate the response of the human visual system, we use a formulation based on a Visible Difference Predictor, VDP [7]. The VDPs predict the per-pixel probability that an observer will detect a difference between two images rendered using different approximations. Using this model of the human visual system, the perceivable error can be written as the product of the VDP times the physical error. We can write the perceived error as:

$$\varepsilon_i = \frac{1}{A_i} \cdot \sum_{v_i \le m < j_i} \left(A_{i,m} \cdot f(m, j_i) \right) \quad (6)$$

where v_i is the lowest mip-map level visible in the current frame for texture T_i (finest resolution used), m is the m-th mip-map level (m is greater than v_i), A_i is the area covered in the gold standard image by texture i, $A_{i,m}$ is the area covered in the gold standard image by m, and $f(m,j_i)$ computes the error using a filtered texture j_i (with a lower resolution) versus a higher resolution one, m. Intuitively the error produced by using j_i is the weighted average of the perceived error in the regions drawn using the different mip-map levels m, this takes into account the fact that if a mip-map level is only used in a small region, the corresponding change in visual quality will be small.
For the levels when $v_i \le m < j_i$ (the ones that need to be drawn but are not in the subpyramid (T_i, j_i)), the error can be estimated as:

$$f(m, j_i) = \frac{1}{A_{i,m}} \cdot \sum_{x,y \in A_{i,m}} \left[\left(\Delta_{images}^{j_i,m} C \right)^{x,y} \cdot VDP^{x,y} \right] \quad (7)$$

Here, $f(m,j_i)$ is equal to the difference Δ in color C using mip-map level j_i instead of m multiplied by the probability of detection of the error *(VDP)*, normalized by the pixel coverage. The differences are taken in the CIELAB perceptually uniform color space.

To compute the probability of detection, we chose the *VDP* presented in [16] for its accuracy and computational efficiency[1].
This *VDP* is defined as:

$$VDP = \frac{1}{TVI \cdot Elevation} \quad (8)$$

Threshold vs. intensity function (*TVI*), describes the luminance sensitivity of the visual system as a function of background luminance, modeling visual adaptation. The *Elevation* term describes the changes in sensitivity caused by the frequency content of the image. It is based on the contrast sensitivity function (*CSF*), which describes variations in sensitivity for patterns of various spatial frequencies, corrected by a masking function to capture the visual system's nonlinear response to pattern contrast. In this formulation, the luminance dependent *TVI* component can be computed in real-time once per frame, but the spatially-dependant *Elevation* component one cannot, which makes this VDP too slow to be used in our system.
The insight that allows us to speed up the evaluation of this metric is the fact that our application does not require a VDP that is accurate for each pixel but only for each texture. Since the most expensive part of the computation is evaluating the spatial contribution *(Elevation)*, we will separate it from the computation of the luminance contribution *(TVI)* by decoupling the two components. By taking this conservative approximation, we obtain a VDP formulation that is efficient enough for real-time applications, while accurately predicting the perceived error for each texture. The function $f(m,j_i)$ becomes:

$$\begin{aligned} f(m, j_i) &= \text{LuminanceContribution} \cdot \text{SpatialContribution} \\ &= \left[\frac{\overline{V}_{i,m} \cdot \overline{T}_{i,m}}{TVI(\overline{V}_{i,m} \cdot \overline{T}_{i,m})} \right] \cdot \left[\frac{1}{n_{texels} \cdot \overline{T}_{i,m}} \cdot \sum_{u,v \in T_{i,m}} \frac{\left(\Delta^{j_i,m}_{texel} T\right)^{u,v}}{Elevation(T_{i,m})^{u,v}} \right] \end{aligned} \quad (9)$$

where $\overline{V_{i,m}}$ is the average of the Gouraud interpolated vertex colors in the area covered by mip-map level *m* of texture *i* and $\overline{T_{i,m}}$ the average color of mip-map level *m* of texture *i*. Note that the differences are taken only on the texture values *T* (and not the color ones *C* as before). Figure 1 illustrates the computation of the spatial contribution. The efficiency of this new metric derives from evaluating the spatial contribution as a pre-process (which can be done during mip-map pyramid creation), and by evaluating the simpler luminance component on the fly.

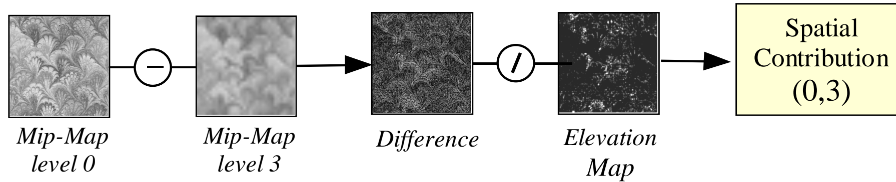

Figure 1: Computation of the spatial contribution of the benefit function.

[1] The reader should consult [16] for implementation details and theoretical justification of the computational model presented.

4. Implementation details

4.1. System overview

We implemented our texture caching algorithm in a real time walkthrough application written using standard OpenGL. The scene vertex colors are produced by a mesh simplification algorithm performed on a tone mapped hierarchical radiosity solution. Any global illumination algorithm can be used to provide these per vertex color values. If no global illumination solution is available, the direct illumination provided by the hardware shading could be used by our metric. The system is composed of off-the-shelf PC components: a Pentium III processor with a GeForce II Ultra graphics board with 64 MB of video RAM.

The resource allocation algorithm runs asynchronously and with lower priority than the display loop, to ensure that the frame rate will not be affected by the execution of the algorithm itself. After the computation of the benefit function (as detailed in the next section), the resource allocation algorithm determines the best possible texture set to use when drawing the scene. It then loads the specified subpyramids in texture memory by using OpenGL texture priorities and loading only textures that are needed but do not reside yet in texture memory. This ensures a minimum number of texture switching operations. We also decided to amortize the total cost by distributing the computation steps over time, especially the switching of textures to avoid large demands on the hardware at the same time. By doing this, the overhead per frame does not exceed a couple of milliseconds.

4.2. Benefit function evaluation

In order to evaluate our benefit function we need to compute the area covered by each mip-map level of each texture and the average color $\overline{V_{i,m}}$ in each of those regions. To compute these values, we need the following per-pixel information: *V*, *textureID* and *mip-map level*. We obtain this information with a single display pass called *TextureIDMap*, where for each polygon, the *R*, *G*, *B* values of the vertices are set as:

$$R_P=V,\ G_P=\text{TextureID},\ B_P=1 \qquad (10)$$

To determine the mip-map levels required when rendering with the high-resolution gold standard texture set, we render the polygons with a special mip-mapped texture, whose values are constant over the texels. This encodes the mip-map pyramid levels:

$$R_T=1,\ G_T=1,\ B_T=\text{mip-map level} \qquad (11)$$

The combination of the polygon color and this texture (with blending operation) fills the frame-buffer with all the required information. After this step, all per-pixel information has been calculated.

Treatment of the frame-buffer values now provides the final information required (e.g. pixel coverage per texture and mip-map level - A_i, and $A_{i,m}$ -, color values $\overline{V_{i,m}}$…). Since the allocation algorithm runs asynchronously, we use a prediction camera placed slightly behind the location of the actual viewing camera to anticipate the appearance of previously non-visible textures.

This *TextureIDMap* predicts mip-map level usage while solving the occlusion problem. If a textured polygon is occluded by another one its texture quality q_i is reduced (eventually to zero). Unlike previous approaches our metric takes occlusion events into account. This allows us to handle both open environments such as terrains and cluttered environments such as architectural walkthroughs. The *TextureIDMap* is illustrated in Fig. 2 (here the contrast of each map has been enhanced to facilitate reading the figure).

5. Results

We tested our algorithm on a highly textured architectural scene containing approximately 100,000 radiosity elements after mesh simplification. Figure 3-A shows the variation over time of the values of the benefit function (q_i), for each texture *i* present in this environment. As the observer moves into the scene, new parts of the scene become visible (or occupy more space on the screen), while others go out of view. As a result, the evaluation of the quality function varies with time. Figure 3-B presents the evolution of the active texture set over time for the same scene during the walkthrough. This graph shows the minimum mip-map levels that are loaded onto the graphics board for each texture present in the scene. As one can observe, in this walkthrough, most of the time the minimum mip-map levels used are 1 or 2 except between the 50th second and 65th second of the walkthrough where dramatic changes in the rendering state occur, since the observer has moved toward a picture on the wall to see it in greater detail. As he moves toward the picture, the resource allocator gives more memory to this particular texture to permit rendering it at full resolution (level 0). As a result, this affects the approximations chosen for the other textures and some of them are almost completely removed from the board memory.

We also tested this scene with high resolution textures (1024 by 1024) to greatly overload the board memory (the set of textures was 4 times bigger than the memory allowed by our system). Under these conditions, the rendering system ran at 3 frames per second. With the framework presented in this paper the framerate remains above 40 frames per second as shown in Figure 3-C.

Note that we also compared our algorithm with a load/unload priority scheme that used a visibility heuristic. In many circumstances this texture management heuristic will fail. Indeed, if in one frame all the visible textures overload the board memory, this heuristic will preserve image quality but at the price of a dramatic reduction in frame rate. In our test we found that in some cases, this method could only produce 4 frames per second while our algorithm yielded rates above 60 frames per second. Progressive loading as in [5] would solve this problem but without any guarantee on image quality.

Some snapshots from the walkthrough scene are shown in Figure 4. In these images the diagram in the left indicates the approximation chosen for each texture (the longer the bar, the coarser the approximation). The graphs indicate that the optimal texture set is different for each location. The image on the lower right shows the scene when the observer has moved toward the picture.

6. Conclusion

In this paper we presented a texture caching algorithm that provides high framerates while maximizing image quality on current commodity graphics hardware. Unlike previous approaches, our algorithm uses an efficient perceptual metric to determine where lower resolution textures can be used without reducing the visual quality of the resulting images. It relies on a textureIDMap to solve the visibility problem and estimate mip-map level usage. The algorithm should be useful in a variety of interactive rendering scenarios and could be incorporated and optimized for graphics APIs like Performer, Direct3d or OpenGL to improve the performance of PC rendering applications.

This texture caching algorithm may be employed on top of main memory and disk based texture management schemes (as in [5]) and is compatible with geometric simplification approaches (LOD, culling, impostors…) used to reduce the demands on the graphics pipeline.
The benefit function that drives our algorithm is of general utility and can be integrated in various other applications. For example, it could be used to prioritize network bandwidth in client/server rendering for telecollaboration or shared virtual environments. It could also be used to automate the laborious hand-tuning of texture and environment maps currently done in memory-constrained console gaming applications by treating main memory as a limited resource.
In the future we would like to develop more advanced perception metrics to further increase the efficiency and effectiveness of our algorithm, for example, by taking the perceptibility of color differences into account. We would also to extend our algorithm to handle additional map-based shading methods (e.g. shadow maps, light maps, environment maps, multitexturing) to increase the realism and performance of interactive rendering applications.

7. Bibliography

[1] A. C. Beers, M. Agrawala and N. Chaddha. *Rendering from Compressed Textures.* SIGGRAPH '96 Conference Proceedings.

[2] J. Blow. *Implementing a Texture Caching System.* Game Developer, April 1998.

[3] M. R. Bolin and G. Meyer. *A Perceptually Based Adaptive Sampling Algorithm.* SIGGRAPH '98 Conference Proceedings.

[4] S. E. Chen. *Quicktime VR – An image-Based Approach to Virtual Environment Navigation.* ACM SIGGRAPH '95 Conference Proceedings.

[5] D. Cline and P. K. Egbert. *Interactive Display of Very Large Textures*. IEEE Visualization '98 Conference Proceedings.

[6] D. Cohen-Or, E. Rich, U. Lerner and V. Shenkar. *A Real-Time Photo-Realistic Visual Flythrough.* IEEE transaction on Visualization and Computer Graphics, 1996.

[7] S. Daly. *The Visual Difference Predictor : An Algorithm for the Assessment of Visual Fidelity*. Digital Image and Human Vision, MIT Press, 1993.

[8] T. A. Funkhouser and C. H. Sequin. *Adaptive Display Algorithms for Interactive Frame Rates During Visualization of Complex Virtual Environments*. SIGGRAPH '93 Conference Proceedings.

[9] I. Homan, M. Eldridge, K. Proudfoot, *Prefetching in a Texture Cache Architecture.* Proc. 1998 Eurographics/Siggraph workshop on graphics hardware

[10] E. Horvitz and J. Lengyel. *Perception, Attention, and Resources: A Decision-Theoretic Approach to Graphics Rendering*. Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence, 1997.

[11] P. Lindstrom, D, Koller, L. F. Hodges, W. Ribarsky, N. Faust and G. Turner. *Level-of-Detail Managements for Real-Time Rendering of Photo-Textured Terrain.* GIT-GVU Technical Report, 1995.

[12] T. Möller and E.Haines. *Real-Time Rendering.* AK Peters, 1999.

[13] K. Myszkowski. *The Visible Differences Predictor : Applications to Global Illumination Problems*. Eurographics Rendering Workshop '98 Proceedings, 1998.

[14] K. Myszkowski, P. Rokita and T. Tawara. *Perceptually-informed accelerated rendering of high quality walkthrough sequences*. Eurographics Rendering Workshop '99 Proceedings, 1999.

[15] S. M. Oborn. *UTAH: The Movie*. Master's Thesis, Utah State University, 1994.

[16] M. Ramasubramanian, S. N. Pattanaik and D. Greenberg. *A Perceptually Based Physical Error Metric for Realistic Image Synthesis*. ACM SIGGRAPH '99 Conference Proceedings.

[17] Sahn. *Approximate algorithms for the 0/1 knapsack problem*. ACM Publication.

[18] *S3TC DirectX 6.0 Standard Texture Compression.* S3 Inc., 1998.

[19] J. Torborg and J. T. Kajiya. *Talisman: Commodity Realtime 3D Graphics for the PC*. SIGGRAPH '96 Conference Proceedings.

[20] H. Yee, S. N. Pattanaik and D. P. Greenberg. *Spatio-Temporal Sensitivity and Visual Attention in Dynamic Environments*, accepted for publication in ACM Transactions on Computer Graphics (2001).

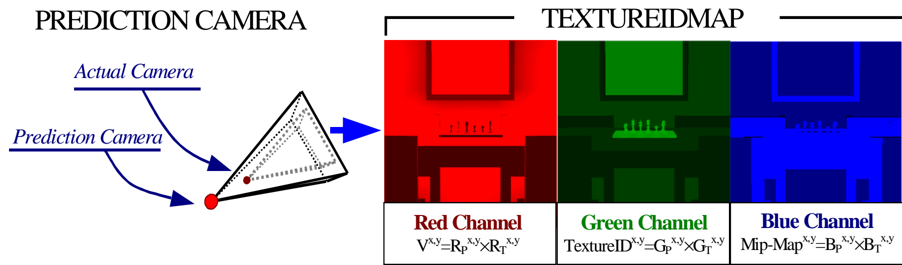

Figure 2: TextureIDMap computation.

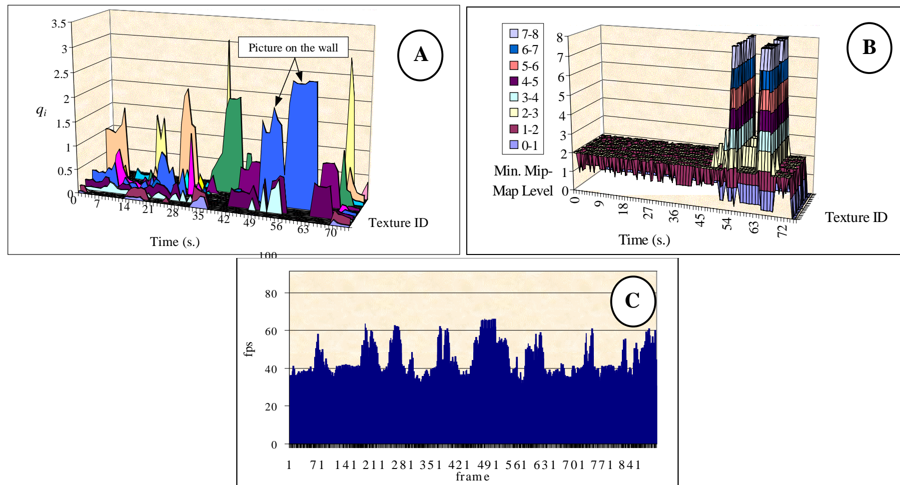

Figure 3: Variation of q_i (A), subpyramids loaded (B), frames per second (C) along time.

Figure 4: Walkthrough of an architectural scene
(performance with our caching strategy is 40 fps, without is 3 fps).