A Realtime Immersive Application with Realistic Lighting: the Parthenon

M. Callieri * P. Debevec ** J. Pair * * * R. Scopigno *

Abstract

Off-line rendering techniques have nowadays reached an astonishing level of realism but pay the cost of long computational times. The new generation of programmable graphic hardware, on the other hand, gives the possibility to implement in realtime some of the visual effects previously available only for cinematographic production. We describe the design of an interactive system which is able to reproduce in realtime one of the crucial sequences from the short movie "The Parthenon" presented at Siggraph 2004. The application is designed to run on a specific immersive reality system, making possible for a user to perceive the virtual environment with a nearly-cinematographic visual quality. In this paper we present the principal ideas of the project, discussing the design issues and the technical solutions used to implement the realtime demo.

Key words: realtime shading, realistic lighting, 3D scanning, immersive system

1 Introduction

Realtime and off-line rendering have always been considered two separate worlds, since it is usually very hard to mix the astonishing level of realism obtained in movies with the user control available in videogames. In the last few years, however, many examples of offline-to-realtime conversion have been presented, mostly due to improvements in video card technology.

A couple of years ago the ICT Graphic Lab of the University of Southern California started a project aimed at performing the 3D acquisition of the Parthenon building and of all its carved decorations with the aim of building up a complete virtual reconstruction of the building in its current and original shape [19]. A collaboration

* Istituto di Scienza e Tecnologie dell'Informazione (ISTI)-CNR, Pisa, Italy
**ICT Graphic Lab, University of Southern California, Marina del Rey, USA
* **ICT FlatWorld, University of Southern California, Marina del Rey, USA

between the Visual Computing Lab (VCLab) of ISTI-CNR and the ICT Graphic Lab started in the early phase of the project, since ICT choose the VCLab tools [2] to process the raw scan data. The most spectacular outcome of the Parthenon Project has been for sure the short movie presented at the Electronic Theatre at Siggraph 2004. The objectives of this work is to reproduce in realtime a crucial sequence of the short movie: the **time lapse** sequence which shows the Parthenon during the passing of a whole day, from dawn till dusk. This sequence shows the interaction between the changing sun and sky against the building geometry, originating complex lighting effects and bringing out the surface details.

The aim of the demo we have designed for an immersive VR platform is to convey the same level of realism of the Sequence computed offline, reproducing the correct interaction between the light and the building and thus giving to the viewer a feeling of presence. Some peculiar characteristics of the work can be summarized as follow:

Complex Shading: the idea behind this work is not to find tricks that can produce "plausible" results, but to find an approach which allows to compute a realistic lighting even in a realtime environment.

HDR: High Dynamic Range calculation is considered essential to convey a sufficient level of realism, especially when we are interested in outdoor scenes. Sun intensity can be over five orders of magnitude brighter than the sky and clouds, giving a dynamic range very difficult to manage. Working on color with only 8-bit-per-channel would result in excessive loss of visual details.

Immersive Stereo: running the demo in an immersive environment will greatly enhance the experience. The VR platform used allows both to manage interaction in a very natural way, reacting to user movements and supporting a very large displaying surface (by retroprojection), rendering the scene at a *scale* much closer to reality.

The use of virtual reality to better present cultural heritage artifacts and environment is quite an old trend [4,16] and cultural heritage applications have been the standard demos for many virtual theaters or CAVE-like systems.More recently, the need for improved realism in this kind of VR applications has been addressed by many research projects which focused on improved geometric models and more sophisticated use of textures [7,20]. But using good geometry and textures is not sufficient to convey a sense of presence, adding a good modelling of the illumination is another basic ingredients [17]. How to add more advanced illumination models in interactive VR environments is still an open research topic.

A requirement of the project was that the interactive visualization should be implemented in the framework of a specific virtual reality environment, called FlatWorld. FlatWorld is a peculiar system, based on retroprojected walls, which simulates a room with openings (virtual windows) over the external world. The need to develop the application for this specific environment introduced some constraints, described in the following sections.

Fig. 1. Available data. The 3D model of the Parthenon building, acquired with TOF scanning and divided in multiple chunks (one of these chunks is rendered in white); the texture atlas used for the highlighted chunk, which encodes the recovered surface albedo, and the texture-mapped chunk; finally, HDR images of the sky dome.

2 Available ingredients

The interactive demo has been designed by using the 3D data gathered during the Parthenon project. Working on the same data used to produce the Siggraph movie was a wonderful opportunity to focus on the rendering techniques, since this data has proven to be accurate and complete, as shown in the Siggraph movie.

Obviously, the core of the interactive demo is the 3D model of the parthenon. This model has been acquired using a *time of flight* scanner and then completed with some elements modeled with standard tools (the latter were needed to complete parts which were hard to be acquired because of occlusions). The model has been divided in multiple chunks during its generation, to help managing the large amount of data.

Another important data for a realistic rendering is the surface characterization, since the knowledge of the reflection properties of each part of the building makes it possible to compute a physically plausible illumination. For the Parthenon dataset, an accurate description of reflection characteristics was available. First of all, the general BRDF of the building surface has been measured. Then, the surface albedo has been recovered for the two short sides (the other two sides of the Parthenon were occluded by the scaffolding installed for restoration purposes), using an inverse rendering method [5].

However, to obtain a realistic illumination of the building during the entire day sequence, it is necessary to have a faithful representation of the light sources; in this outdoor scene, the two major sources of light are the sun and the rest of the skydome. A day-long sky dataset has been acquired using HDR imaging techniques [18], where a High Dynamic Range skydome image is available for every minute in the day. Moreover, for each image, the sun position and intensity have been detected, producing a concise directional light representation for each minute of the day. This dataset has been used in the movie rendering to illuminate the scene; the sun is the major light source but, to obtain a realistic illumination, also the rest of the skydome has been taken in account.

To render the Parthenon movie with a high level of realism, a physically accurate

rendering engine is needed. The Arnold rendering engine [8] was used to render The Parthenon movie. It implements a Monte Carlo-based global illumination algorithm able to produce images with a very high realism. Moreover, coming as a library, it is completely configurable and it is possible to add plugins to manage custom data or to write shaders which implement new surface behaviors. Obviously, it was not possible to directly use the same engine in a realtime system because of the high computational cost; to render the movie, one hour was necessary to compute each frame. However, this rendering engine can be used to calculate *part* of the shading equation and the results can be used in the realtime demo.

A design constrain for the interactive demo was that it should run on the FlatWorld System [14,13], a virtual reality environment focused on training. In film and theatrical productions, sets are constructed using modular components called *flats*. FlatWorld utilizes a *digital flat* system. A digital flat is basically a large retroprojective display; the actual setup is a room with two active walls and the user it's free to move inside this room, looking at an outside world displayed on those walls.

3 Design of the Rendering Method

Our interactive demo should represent both the **direct** and **indirect** illumination: the main effect of direct illumination is the production of hard shadows and base lighting, while indirect illumination smoothes up the effect introducing additional light bounces and taking in account light coming from all the skydome. Since the different nature of the two components we decided to employ two different algorithms for the computation: one able to generate precise shadows, the other able to produce diffuse effects.

The mostly static nature of the scene suggested that the best approach was to precalculate as much as possible the invariants in the illumination equations and just complete the calculation at runtime when all data is available. Hardware shaders seemed to be the perfect choice for this kind of task since it is possible to implement efficiently the lighting calculation. Moreover, the processing power of GPUs will steadily increase in the immediate future (faster than CPU grow), providing a better frame rate and the possibility to add more complex effects. Precomputing shadows is also a good way to limit the geometry to be drawn each frame: using shadow maps or shadow volumes would require all the occluding geometry to be drawn (even the non directly visible parts); obviously, a subsampled version of the occluders can be used but this would affect shadow precision. Conversely, by precomputing shadows we use the whole dataset for shadow calculation, but only a small part of the geometry will be rendered and shaded by the realtime application. The same holds also for indirect illumination, since light bouncing between geometric elements is determined using the whole dataset, but only the data related to the realtime model will be stored and processed at runtime.

The direct lighting algorithm should firstly discriminate between *shadowed* and *lit*

areas, this has been done with a variation of the **interval mapping** technique (described in the *GPU Gems* book [9]). Dynamic shadows are normally a difficult task, but in this case the only direct light source in the scene is the sun that is a *directional* light, easier to manage with respect to a local point source; moreover, its location in every minute of the day is well known. It is therefore possible to regularly sample the daytime and to calculate the shadows for each of the sun position in the timeline. There is no need of storing all the shadows in the scene, since the only rendered shadows will be the ones casted onto the visible geometry. The idea is to encode for each part of visible geometry which are the *time steps* that geometry will be illuminated directly by the sun or, conversely, the ones it will be covered by shadows. At runtime, each vertex will be tested to know its state with respect to the current time. Lit areas will be illuminated using a Lambertian illumination calculation, which helps simplifying the computation without affecting too much the final visual quality, since the measured Parthenon BRDF proved to be quite close to a Lambertian surface.

The diffuse calculation is based on **spherical harmonics** [10]. Spherical harmonics basis are a way to encode a signal over a sphere and in our case the signal to be encoded is the sky image probe. The initial step is to calculate for each part of the geometry the lighting contribution from a particular harmonic; then the lighting condition is encoded using the harmonic basis. To compute the lighting at realtime, for each part of the geometry the influence coefficients are multiplied by the sky dome encoding for that particular time position and added up. Spherical harmonic lighting is nowadays a very popular technique to calculate lighting [11,15]; the algorithm used in the our interactive system is a standard implementation

The two light contributions are added up obtaining the amount of light reflected by the surface. This value is then modulated by the surface albedo to obtain the color value of that surface point.

4 Hardware Implementation of the Shading Process

All the lighting calculation are based on vertices and this choice is justified by the following arguments. The model has a very high resolution, so lighting calculated just on the vertices produces high quality results; then, given the very complicated model topology, it's very hard to produce a good (and compact) texture parameterization.

As stated before, the lighting process is divided in two steps; firstly the direct lighting component will be calculated, then the indirect contribution will be added. As introduced in the previous section, sun direction and intensity is know at each time of the day; to compute *direct lighting* on a surface point it's only necessary to know if that vertex receives light or not in the specific time. Therefore, together with each object vertex we store a bit mask containing a bit for each time step in the day, representing the lighting condition (1 for light or 0 for shadow) of that

particular vertex in that particular time. Lighting is therefore:

$L_{direct} = Lambertian$ if mask[current_time] = true
$L_{direct} = 0$ if mask[current_time] = false

For *indirect illumination*, the light contribution in each vertex is the sum for each harmonic of the influence coefficient (how much the vertex is affected by that harmonic) multiplied by the current sky encoding (how intense the sky is on that harmonic).

$L_{indirect} = \sum_{i=1}^{4} Coeff_i * SkySH_i$

The shading algorithm has been implemented firstly using **OpenGL Shading Language**, for testing the algorithm in a sample program and then converted to **HLSL** to be used in the GameBryo engine; the conversion is quite straightforward and involves almost only changing names of intrinsic functions and data types.

Hard shadowing is the first step of the lighting process. For each vertex is necessary to know if in the current time that point is in light or if it is under shadow. The status of the vertex is determined by accessing the shadow mask ant testing the appropriate bit. The binary mask is implemented using floating values, because floating points variables are native in graphics hardware while integer (normally used as bitmask) are just emulated. Each float (in the standard IEEE implementation) has a 23 bit mantissa that can be used to store 23 binary samples; the mask is composed by 4 float, giving a total of 92 samples. More sample could be stored in a single float (for example, using the float sign) but that will cause problems in the hardware shader since it would be necessary to discriminate particular cases. According to our specific sky dataset, the first time the sun is visible is at 6:52 and it remains visible until 17:44. Using the 92 samples encodable in the four floats it is possible to cover the time from 6:57 to 17:34 having 7 minutes lapse between each sample. Each vertex has 4 floating point values containing the shadow mask; since each float contains 23 samples the bit we need can be found as bit `N1` of float `N2` with `N1 = CurrentTimeIndex mod 23` and `N2 = CurrentTimeIndex div 23`. To avoid calculating those values for every vertex the two indexes are computed in the program and passed down as constants. Binary operations are not implemented into shaders, therefore the old trick of "divide and check the rest" is used to extract the bit value: `((mask[N2] / 2*N1) mod 2)` returns 1 if the bit is set, 0 otherwise. After this step the light value for the vertex is set to 0 (if in shadow) or to the standard Lambertian lighting value (if in light) calculated as:

```
DirectL = LightIntensity * max(dot(VertexNormal,LightDir), 0.0f)
```

Given the direct lighting contribution, we have to add the diffuse component computed using the spherical harmonics. Each vertex has 4 float which represent the amount of influence the spherical harmonic base has on that vertex; this is used as a multiplication factor for the sky probes encoded as spherical harmonics. The lighting contribution of each harmonic to a particular vertex is obtained by multiplying the influence factor by the spherical harmonics encoding of the sky (basically, an

HDR RGB value) and summing it up to the direct lighting previously calculated:

```
Directlight.r += vert_sh[0]  *  sky_sh_encode[0].r;
Directlight.g += vert_sh[0]  *  sky_sh_encode[0].g;
Directlight.b += vert_sh[0]  *  sky_sh_encode[0].b;
. . . . . [omissis harmonics 1-2-3] . . . . . .
```

Only the first four harmonics are used into our interactive system; in other publications nine harmonics are used to obtain a more precise representation of the high frequency shading variations. However, in our case, we are using this kind of technique to calculate just *half* of the lighting; at this point the high frequency direct illumination is already calculated and we just need introduce low frequency indirect lighting.

At the end of this process we have the final light value incoming in that vertex. This value is then passed down toward the fragment shader. In this way the lighting values calculated on the vertices are interpolated across the faces of the model; this produces a smooth and plausible effect, under the hypothesis that the base geometry is tessellated with sufficient density and that the user will never approach too close to the geometry. In our case the geometry is dense and the usual view specs makes each triangle project on 3-4 pixels area on the FlatWorld display. Using a less detailed model, would require moving the computation from the vertex to the pixel/fragment, therefore requiring to encode all the needed parameters on a texture map (and thus a parameterization will be needed).

In the fragment shader the light value is multiplied by the surface albedo, producing the final color of the fragment. Up to this point all the computation has been performed using floating point values, it's now time to get from HDR to values in the [0..1] interval for the final rendering. The color is multiplied by the exposure level (calculated as 2^{stops}) coming as a shader constant from the application and clamped in the [0..1] interval. Gamma correction is then applied to match the monitor (or videoprojector) response.

5 Offline Parameters Calculation

As stated before, the main idea behind the interactive system is to precalculate all the invariants: all the shading computation is based on different parameters that have to be computed offline. This parameters can be divided in two groups: *vertex attributes* and *lapse attributes*. Vertex attributes are stored in each vertex of the 3D model and used during computation, they appear as variables in the vertex (and fragment) programs since they are model-dependant. Lapse attributes represent the light condition in a particular time of the day and they are generated from the skydome probes.

To precalculate the invariants of lighting equation, it is necessary to have an implementation of the lighting process that can be used to compute lighting "up to" a

Fig. 2. Examples of real-time shading; note how the shadows moves precisely over the geometry (top-let and bottom-left images) and how the HDR lighting calculation allows to adjust the exposure to better percieve details (images on the right) which are under shadows in the images on the left.

specific point and then retrieve the partial result. The Arnold rendering engine is the tool used to produce "The Parthenon movie; it is completely configurable since it is basically a library of functions. To render the offline movie, custom programs have been implemented to manage the data format used for the Parthenon model, the material shaders and the lighting environment. Beside its accuracy, another important feature of the Arnold engine is that it can be used to produce just the shading of a particular point on the scene geometry. Using the Arnold library is therefore possible to build a program that, given in input the scene, computes for each vertex of the geometry the various values we are interested in. The use of the Arnold engine to precalculate the lighting values is not only justified by a code reuse policy (no need to rewrite complex lighting computations) but it also guarantees that the shading results will be coherent with respect to the ones presented in the offline movie.

5.1 Illumination parameters

Two different kind of values are required to compute the direct light contribution: the shadow mask and the sun position and intensity.

Shadow mask. The mask represent the lighting state of the vertex during the day. To calculate these values (for each vertex and for each sun position), the Arnold rendering engine has been used in "probing" mode. The scene with all the geometry has been initialized as doing a normal rendering, then for each sun position we calculated the irradiance of each vertex in the final model using a single ray with no light bouncing; vertices with an irradiance equals to 0 are in shadows. As an alternative, any ray tracing implementation would work fine since

Fig. 3. Shadows precalculation: for each sun position, the scene is rendered in the Arnold Engine. In this case the engine works like a ray tracer, to determine if each single vertex is receiving light in that time of the day.

in this phase it's only important to know if the vertex can see the light source or not.

Sun position and intensity. This is the most intuitive data and comes directly from the processing of the HDR sky probes. In each skydome image, the sun has been detected as the brightest light source; its position has been determined as the centroid of the brightest pixels, the various position during the day have been fitted to a curve to reduce position detection errors due to clouds occlusion. The sun intensity (and color) has been determined as the mean value of the brightest pixels. To have more details on this process, refer to the HDR sky acquisition paper [18]. For each time position there are 3 float to encode the normalized direction [XYZ], and 3 float for the high dynamic range color [RGB].

Again, two different values are required to compute the indirect light contribution: the spherical harmonics vertex response and skydome spherical harmonics encoding.

Spherical harmonics response. Spherical harmonic basis describes a signal over a sphere; using that signal as a skydome light source to illuminate an object it is possible to measure how much each vertex is affected by the lighting. This value, a float for each basis, is used to modulate the spherical harmonic encoding of the real skydome. Again, the Arnold engine in probing mode was used, rendering a scene with no direct light sources and using the spherical harmonics as skydome. This time, light scattering has been included in the computation, using four level of recursion. For each vertex in the model, 4 signed floats are generated.

Skydome spherical harmonics (SH) encoding this value is a representation of the sky using the first four SH basis. In theory it is the sphere integral of the multiplication of the sky by the spherical harmonic; practically it is computed as the weighted sum for all pixel of the multiplication of the sky probe image by the spherical harmonic image. For each harmonic basis there are 3 floats, basically a signed HDR color value.

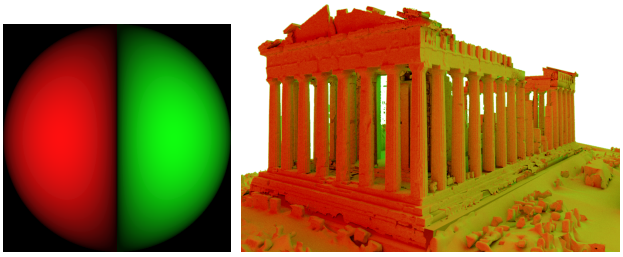

Fig. 4. Diffuse lighting precalculation. On left, the Spherical Harmonic base used for lighting with positive and negative values encoded in red and green. On the right, by illuminating the scene using the harmonic as a skydome light source it is possible to see how much each part of the object is influenced by this harmonic. (beware: the contrast of the right image has been raised to better perceive the differences).

5.2 Geometric Model Setup

The original Parthenon model was too big to be used directly into a real time application: even at a medium resolution the polygon number exceed 10 million triangles. A smaller model was required to grant a realtime frame rate. The 3D model used for the rendering of the movie is composed by different elements:

- the *ground*, obtained by 3D scanning, stored as a single file and textured;
- the *temple*, obtained by 3D scanning, which is divided in various chunks (corresponding to a regular voxel-based space subdivision); only the front and rear facades are textured;
- *filling geometry*, modeled with a CAD system to fill gaps in the scanned data, encoded with a single file.

Since the image resolution available in the Flatworld system is 1024x768 for each wall, an over-detailed geometry will be useless. Moreover, the Flatworld goal is to simulate a *window* over a real environment trough the walls of the visualization room. The room has to be statically placed into the scene and the user will look out from one *virtual* window, having the possibility to move inside the room. The Flatworld system has been designed for military simulation and training applications, such as the ones where a soldier is in a building and observe the external environment (buildings, other actors). This restricts the set of possible view position and directions that the tracked user can generate by moving in the interior space. Therefore, it is not necessary to have a model which should look good from all possible viewpoints, like in a standard browsing/manipulation framework, but just a model that looks good from the viable viewpoints. For this reasons our first task was to choose a position for the virtual Flatworld room (on top of the Acropolis), in such a manner that it should have been possible to see from the virtual window the whole Parthenon at an adequate distance and from a sufficiently low position, to convey a sense of greatness. Part of the sky and of the ground should be visible too to show

the changing of the day and the shadows movement. We choosed a position on the west side, offset towards north, looking more or less at the same area covered by the movie; this also because this is the area with a better 3D scanning coverage.

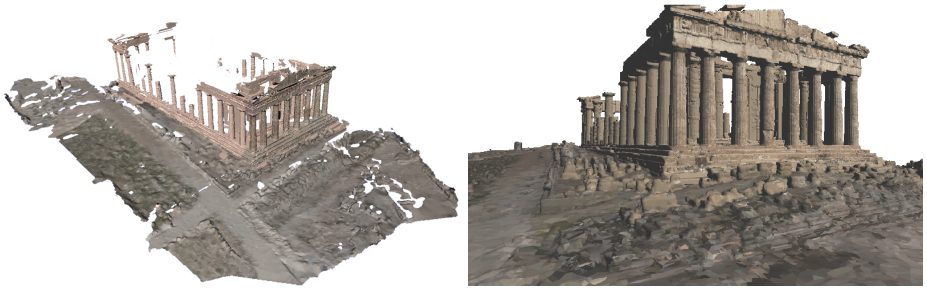

Fig. 5. The 3D model after reduction and culling; even if the model is not complete (left) it contains all the elements that are visible from the FlatWorld environment (right, the view from the center of the room).

The Parthenon model has then been divided in different areas, according to the distance from the room position, the geometry chunks contained in each part have been merged together using different resolutions, to have a high detailed geometry in front of the camera and a less detailed triangulation in the parts farther away. We adopted this static LOD approach since the specific characteristics of the virtual reality environment, Flatworld, did not allowed us to use more efficient and sophisticated multiresolution approaches [3]. The filling geometry, modeled in Maya to close the holes in the scanned geometry, required also some modifications. This added geometry was good for offline rendering, but problematic for our shading algorithms because it is composed by very large triangles and self-intersecting parts. For this reason, the geometry has been recursively splitted into smaller triangles to have a more uniform triangulation and the redundant and self-intersecting parts have been eliminated. These processing steps were needed due to the characteristics of the 3D model. Scanned models are very accurate, but at the same they are usually not topologically clean and highly incomplete, requiring an intense processing.

At this point we had a simplified model, however this model still had too much geometry (2 Million triangles) for our realtime constraints. By exploiting the characteristics of our constrained virtual window system (the model will be viewed from a virtual window and with a freedom of movements of a few cubic meters) we further reduced the size of the geometry by purging those model faces which will always be back-facing or occluded from all possible points of view in the Flatworld space. A ray tracing algorithm has been used to detect those back-facing or occluded faces. The size of the model was reduced to 700,000 triangles, low enough to be rendered in a real time context but still enough to convey a very precise geometry when displayed on the Flatworld screen.

6 Interactive system implementation

Building a single task interactive application, with the possibility to use every bit of the host machine and without coding restriction, can produce very good results, but also the code implemented will not be usable in conjunction with other tasks or inside more structured programs. In our case, part of the initial specifications was the need to integrate the demo in the FlatWorld framework. The idea was to implement the demo as some sort of architectural "stage" in a way such that, afterwards, more elements (people, animals, special effects) could have been added.

Beside its hardware setup, the core of the FlatWorld system is an application framework able to manage the various elements of the immersive system: head tracking system, sound, lighting and so on. This framework is based on a commercial DirectX based game engine called GameBryo [12]. The skeleton of the demo program was already available: a simple application able to render a scene graph and accept input from keyboards, game controllers and the tracking system. The real problem was to integrate all the data and processing requested in a way that was suitable using the GameBryo engine, efficient and still open to future extensions.

Geometry. The geometry has been imported in the GameBryo engine using multiple OBJ file, since it is impossible to have such a large object in a single GameBryo node. This because each geometric node is converted in an high performance indexed buffer on the video card memory and, due to hardware limitations, the size of those buffers is limited to 20k triangles.

Shader data. The shader data is stored in the video memory, in the same objects where geometry is stored. All data is loaded at startup and transferred to the video card; in this way no further updates are necessary at runtime, all animation is done just changing the shader parameters.

Sky data. The sky behind the Parthenon is a rough sphere geometry textured using the sky image dataset. Even if 92 samples are acceptable to show the shadows moving on the objects, for the sky this is quite a poor time resolution since the clouds move much faster than the sun. For this reason, in the FlatWorld demo we used 184 sky maps (two for each time position); this means that during time flow there are two kind of transitions. In odd transition it's only the sky that changes, in even transition the sky and the shadow changes.

Beside pure rendering, the main activity inside the application is to change the time-dependant data, updating the shaders with the new frame constants; this is done by a class called TimeLapseNavigator. The lapse navigation object contains the lapse attributes (sun intensity and position, SH sky encoding, exposures) for each time position. When the time position is changed (automatically or by user command) the shader frame constants are updated using the corresponding data.

Interaction with the demo is quite simple: at the start the viewpoint is set to the center of the virtual room and can be modified using a keyboard, a game controller

or using the tracking system. The demo starts in automatic time flow mode, the time advance automatically throughout the day lapse, completing a cycle in more or less 30 seconds and then starting again (the speed can be specified in the configuration file). This behavior can be interrupted (and then resumed), stopping the demo in a particular time position; time position can be moved forward and backward with just a gesture (using a floating wireless mouse). Exposure is automatically adjusted to a "good" level in the automatic mode but can be tuned up or down with another gesture.

Fig. 6. Screenshots of the Realtime Demo. Time flows from dawn to dusk, shadows moves across the building and the overall hue of the scene changes according to the sky illumination. The shots are taken from different positions, each time nearer to the building.

The current system implementation, despite the complex lighting calculation, resulted quite fast; in stereo, with head tracking, it's able to run at 12 frame per seconds (or at 24 fps on a single monitor). The host machine is a Pentium4 3Ghz with 1GB of ram and a GeForce FX6800 Ultra; the most important part of the machine is obviously the video card, since is where all shading computation is done. The memory footprint of the demo is only 200 MB and the CPU is not completely saturated by the application; this makes if possible to add more elements to the scene, such as dynamic actors.

Regarding the resulting realism, even if the lighting calculation implemented in the demo is just an approximation, many of the lighting effects that appeared on the offline movie are still visible. The shadows are precisely represented, the sky color (yellow in the morning and red on evening) influences the overall hue of the building very realistically and the HDR rendering gives the user the possibility to change

the exposure to bring out different range of details.

7 Conclusions and Future Work

In this paper we presented the development of a realtime version of the time lapse sequence from the short movie *The Parthenon*. We discussed some technical problems related to the size of the dataset and complexity of the lighting computations required; to solve this problems, we presented methods to make a scanned 3D model more adequate for realtime rendering applications and we described lighting algorithms based on the separation between direct/indirect light and extensive precalculation. We showed how to use existing rendering engine to precalculate lighting invariants and how to implement those shading algorithms using modern GPUs. The resulting techniques have proven to be accurate (in terms of rendering results) and affordable (in terms of time) for realtime applications. Moreover, being the algorithm execution restricted to hardware shaders, we showed how this computation has been integrated inside an existing application framework.

Some experiments have been devoted to add to the demo some HDR related effect, e.g. glow, but without much success. Beside pure aesthetical consideration, the time required to compute this effect was too much with respect to the obtained visual impact. Since the techniques used in the demo are quite general, the same kind of computation could be integrated in other existing visualization systems. Having all the calculation done on hardware shaders of modern GPU, it is simple to extend existing rendering engine to accommodate additional data and shader management. In this way it would be possible to add realistic lighting even to large 3D dataset visualization tools.

Thanks
I would like to thanks people that made possible this jointed work, among whom I am pleased to mention Paul Debevec and Diane Piepol. Thanks also to all people at the ICT Graphic lab and Flatworld for their contributions in the development of this work.

References

[1] Tomas Akenine-Mller and Eric Haines. *Real-Time Rendering*. A.K. Peters Ltd., 2002.

[2] M. Callieri, P. Cignoni, F. Ganovelli, C. Montani, P. Pingi, and R. Scopigno. VCLab's tools for 3D range data processing. In A. Chalmers D. Arnold and F. Niccolucci, editors, *VAST 2003*, pages 13–22, Bighton, UK, Nov. 5-7 2003. Eurographics.

[3] P. Cignoni, F. Ganovelli, E. Gobbetti, F. Marton, F. Ponchio, and R. Scopigno. Adaptive tetrapuzzles: Efficient out-of-core construction and visualization of gigantic multiresolution polygonal models. *ACM Trans. on Graphics (SIGGRAPH 2004)*, 23(3):796–803, 2004.

[4] Pape Dave, Tomoko Imai, Josephine Anstey, Maria Roussou, and Tom DeFanti. XP: An authoring system for immersive art exhibitions. In *Proceedings Fourth International Conference on Virtual Systems and Multimedia*, Gifu, Japan, November 1998.

[5] P. Debevec, C. Tchou, A. Gardner, T. Hawkins, C. Poullis, J. Stumpfel, A. Jones, N. Yun, P. Einarsson, T. Lundgren, P. Martinez, and M. Fajardo. Estimating surface reflectance properties of a complex scene under captured natural illumination. *ACM Transactions on Graphics*, Oct 2004.

[6] P. E. Debevec and J. Malik. Recovering high dynamic range radiance maps from photographs. In Turner Whitted, editor, *Comp. Graph. Proc., Annual Conf. Series (SIGGRAPH 97)*, pages 369–378. ACM SIGGRAPH, Aug. 1997.

[7] G. Drettakis, M. Roussou, N. Tsingos, A. Reche, and E. Gallo. Image-based techniques for the creation and display of photorealistic interactive virtual environments. In *Eurographics Symposium on Virtual Environments*, pages 157–166. EUROGRAPHICS, 2004.

[8] Marcos Fajardo. Monte carlo ray tracing in action. In *Course 29*. ACM SIGGRAPH, 2001.

[9] Randima Fernando, editor. *GPU Gems*. Addison-Wesley, 2004.

[10] Robin Green. Spherical harmonic lighting: The gritty details. Sony Computer Entertainment America, Pdf document, 2003. http://www.research.scea.com/-gdc2003/spherical-harmonic-lighting.pdf.

[11] Jan Kautz, Peter-Pike Sloan, and John Snyder. Fast, arbitrary BRDF shading for low-frequency lighting using spherical harmonics. In *In Proceedings of the 13th Eurographics workshop on Rendering*, page 291296. Eurographics Association, June 2002.

[12] Numerical Design Limited. GameBryo game engine. www.ndl.com, 2005.

[13] J. Pair, U. Neumann, D. Piepol, and W. Swartout. FlatWorld: Combining hollywood set-design techniques with VR. *IEEE Computer Graphics and Applications*, January/February 2003.

[14] J. Pair and D. Piepol. FlatWorld: A mixed reality environment for education and training. In *International Conference on Information Systems, Analysis and Synthesis*. SCI/ISAS, 2002.

[15] R. Ramamoorthi and P. Hanrahan. An Efficient Representation for Irradiance Environment Maps. In *Computer Graphics (ACM SIGGRAPH '01 Proceedings)*, pages 497–500, 2001.

[16] P. Reilly. Towards a virtual archaeology. In *Computer Applications in Archaeology*, pages 133–139. British Archaeological reports (Int. Series 565), 1990.

[17] I. Roussos and A. Chalmers. High fidelity lighting of Knossos. In A. Chalmers D. Arnold and F. Niccolucci, editors, *VAST 2003*, pages 195–201, Bighton, UK, Nov. 5-7 2003. Eurographics.

[18] Jessi Stumpfel, Andrew Jones, Andreas Wenger, Chris Tchou, Tim Hawkins, and Paul Debevec. Direct HDR capture of the sun and sky. In *In Proceedings of the AFRIGRAPH*, 2004.

[19] Jessi Stumpfel, Chris Tchou, Tim Hawkins, Philippe Martinez, Brian Emerson, Marc Brownlow, Andrew Jones, Nathan Yun, and Paul Debevec. Digital reunification of the Parthenon and its sculptures. In *4th International Symposium on Virtual Reality, Archaeology and Intelligent Cultural Heritage*. VAST, 2003.

[20] Julian Willmott, Lloyd Wright, David Arnold, and Andrew Day. Rendering of large and complex urban environments for real time heritage reconstructions. In *International Symposium on Virtual Reality, Archaeology and Intelligent Cultural Heritage*, 2001.