Real-Time Reflection on Moving Vehicles in Urban Environments

Alexandre Meyer, Céline Loscos

Real-Time Reflection on Moving Vehicles in Urban Environments

Alexandre Meyer
a.meyer@cs.ucl.ac.uk

Céline Loscos
c.loscos@cs.ucl.ac.uk

University College London
Computer Science Department
Gower Street, WC1E 6BT London, UK

ABSTRACT

In the context of virtual reality, the simulation of complex environments with many animated objects is becoming more and more common. Virtual reality applications have always promoted the development of new efficient algorithms and image-based rendering techniques for real-time interaction. In this paper, we propose a technique which allows the real-time simulation in a city of the reflections of static geometry (*eg.* building) on specular dynamic objects (vehicles). For this, we introduce the idea of **multiple environment maps**. We pre-compute a set of reference environment maps at strategic positions in the scene, that are used at run time and for each visible dynamic object, to compute local environment maps by resampling images. To efficiently manage a small number of reference environment maps, compared to the scene dimension, for each vertex of the reconstructed environment we perform a ray tracing in a heightfield representation of the scene. We control the frame rate by adaptative reconstruction of environment maps. We have implemented this approach, and the results show that it is efficient and scalable to many dynamic objects while maintaining interactive frame rates.

Categories and Subject Descriptors

I.3.7 [**Computer Graphics**]: Three-Dimensional Graphics and Realism.Color, shading, shadowing, and texture.

Keywords

Environment Maps, Image-Based Rendering, Ray Tracing, Real-Time Rendering.

1. INTRODUCTION

In the context of mixed-reality, content often comes from different sources mixing real data sources with virtual ones. When merging the information, one needs to make sure that the added virtual information stays consistent with the rest of the scene. Often, the information coming from real sources looks highly realistic as the 3D model can be mapped with the textures extracted directly from pictures. In the European project CREATE[1], we test a constructivist approach (learning through activity) in the context of cultural heritage and urban planning. A typical scenario for the urban planning application is to simulate a new configuration of a city on immersive displays.

Figure 1: Simulation on immersive displays of a new configuration of a city, here the "place Massena" in Nice where a tramway is added. This image was generated by REVES/INRIA Sophia-Antipolis (http://www-sop.inria.fr/reves), in the context of the CREATE project.

The real environment is captured and built from photographs [25] with additional post-processing to finalize the lighting consistency. Additional features are then added to com-

[1]CREATE is a 3-year RTD project funded by the 5th Framework Information Society Technologies (IST) Programme of the European Union (EU). The official contract with the EU was signed under the contract number IST-2001-34231. The project started in March 2002. www.cs.ucl.ac.uk/create/

plement the simulation: crowd simulation, car simulation, tramway insertion, new placement of pedestrian areas, pedestrian crossings, trees, etc. The system will be used by urban professionals to understand and visualize changes and also by the general public to whom it is important to sell the feasibility and the benefits of modifying some areas. It is therefore important for the simulation to be believable, and as photorealistic as possible. In figure 1, the reconstructed model from a real scene is shown, together with the insertion of virtual trees and a tramway. Shadows of virtual elements onto the reconstructed model are simulated. However, no reflections were computed on the tramway making it easy to recognize it as computer graphics rendering.

Lighting effects ranging from shadows, direct illumination, to global illumination as well as transparency and reflection/refraction, have been shown to be extremely important for realism in certain applications [24], giving to the user valuable cues on the environment, about where objects are physically placed, and their material properties, making it easier to identify them. However, the computation of these effects is often restricted to very limited local areas and too few dynamic (moving) objects. In our mixed-reality application, the available computation time is even smaller as it needs to be shared by the display of the static model, the various dynamic objects, their simulation, the sound simulation and the haptic interaction. To be general, our main goal is thus to increase the photo-realism in real-time computer graphics applications with no restriction on the geometry and the number of dynamic objects.

In this paper, we concentrate on simulating reflections of the environment onto animated, specular virtual objects moving in a structured complex virtual environment: vehicles driving in a city. Often, environment maps (EMs) [4, 12, 32] are used for real-time applications instead of computationally intense algorithms such as ray tracing. The concept of EM techniques is to store images of the environment onto a simple geometric object (a sphere or a cube) and to use this information as a texture for computing the reflections onto an object, instead of using the geometry of the scene. Since environment maps are valid for one particular point or its surrounding, the case of moving objects far from this point forces an expensive re-computation of the EM for each object's new position. In the case of environment cube maps it corresponds to rendering the scene six times per object, once for each face of the cube. Currently, to maintain a constant frame rate, only a few moving objects are allowed (usually one).

We propose a new technique, called *multiple environment maps* or MEM, making use of the advantages of image-based approaches, to interactively compute individual environment maps around each moving object, thus obtaining realistic reflections with a quality comparable to ray-traced rendering. Our approach is simple and needs a classical graphic hardware since the most advanced feature we use is the well-supported cube map and multi-texturing extension.

In section 2, we briefly introduce previous work relevant to our method. We then summarize our contributions by giving an overview of the MEM in section 3. In section 4, we describe the techniques used for each of the steps of the method. Section 5 describes our levels of detail and we, then present some implementation and discussion in section 6 and results in section 7, and then conclude.

2. PREVIOUS WORK

2.1 Environment Maps

Accurate reflections between arbitrary objects can be produced by ray tracing [33], even at interactive frame rates with a cluster of PCs [23]. Nevertheless, with a single computer with a common graphic board the cost is significantly higher than texture-mapped polygon rendering, and thus it is fair to say that ray tracing is still difficult to use for real-time applications.

Environment maps are heavily used nowadays in computer graphics. They were introduced to approximate reflections for interactive rendering [4, 12, 32]. But they can be used as a means to render glossy reflections [20, 6, 15, 16] by pre-filtering a map with a fixed reflection model or a BRDF (bidirectional reflectance distribution function). Recent graphics card extensions support the use of environment maps in realtime. Graphics cards now support commonly cube maps [22], and last year, ATI [3] presented at SIGGRAPH a real-time demonstration of high-resolution normal maps to illustrate the importance of normal precision in this technique.

Attempts have been made to use environment maps for less constrained navigation than the one described previously. Quicktime VR [8] is a pure image-based technique, which uses a series of captured environment maps to allow the user to look around a scene from fixed points in space. Cabral *et al.* [6] store a collection of view-dependent EMs. Hakura *et al.* [13, 14] tackle the problem of transition between EMs. Our framework is similar to these methods,however we differ in two different aspects. First, in the way we capture the EM. Their parameterized EM is a sequence of EMs recorded over a dense set of viewpoints whereas our technique needs a less dense set of pre-computed EMs. Second, to recompute an intermediate EM, they represent reflected geometry by simple geometry, such as a box, a parallelepiped, a sphere, an ellipsoids. In contrast, we represent the environment by an accurate representation of the geometry.

2.2 Computing New Views

The problem of computing an intermediate view starting from pre-computed is well known in image-based rendering. Only few papers [13, 14] apply this to EM. The spectrum of image-based methods to solve this problem ranges from those that exclusively use images to those that re-project acquired imagery onto geometric models.

Quicktime VR [8] offers a realistic view of the scene, but the user has a correct perspective from a set of specific locations. Image morphing approaches [7, 27] allow some range of motion. However, the transformation of the reference views to the desired views is *approximated* by interpolation. To maintain a high frame rate, the Talisman architecture [31] reused portions of rendered images and re-project them onto a new view using 2D warping. The Lumigraph [11] and Light Field [17] densely sample light rays to create a ray database. Unfortunately, these techniques are not designed to manage objects with very different scales as we have in our case.

At the other end of the spectrum are methods based largely on geometry. Texture mapping is the most common way to incorporate images into the scene description. Debevec *et al.* [9] represent the scene with an approximate geometric model (semi-automatically created) texture-mapped, in a view dependent fashion, from photographs.

McMillan and Bishop's [19] and Shade *et al.* [28] methods are in the middle of the spectrum, representing the scene as a collection of images that in addition to color also store depth at each pixel. The desired view is generated by a 3D-warp of the depth images. Buehler *et al.* in [5] generalize both the Lumigraph and texture mapping approaches. Their algorithm is defined to work on a wide range of differing inputs, from few images with an accurate geometric model to many images with minimal geometric information.

We differ from texture-based approaches in the sense that we do not re-project all pre-computed images, nor all geometry. Indeed, with the help of a geometry representation we find which parts of the pre-computed images can be used. Moreover, because our algorithm is based on this geometry, it minimizes the size of the stored data, which makes it more accurate than morphing techniques.

3. OVERVIEW

Since re-rendering the whole scene on a new EM around each moving objects makes it impossible for real-time (See table 1), we propose a new method based on pre-computed EMs to interactively, on-the-fly reconstruct these new EMs. We place around the environment a set of *reference environment maps* or REMs, pre-computed before the simulation. Each of these REMs stores information both on colors and on the depth from the static surrounding environment. The positions of the pre-computed REMs define in the scene a non-constraining path, where the objects will be able to move. The current navigation algorithm allows objects to move away from this area, however there is no guarantee that the information provided by the REMs will be sufficient for the computation of the reflections onto these objects. Our paper does not address the problem of positioning the REMs. We assume that the REMs are set around the scene, so that a maximum of visible objects are captured in the area where dynamic reflective objects move. Several papers tackle this problem [10, 29, 1].

At run time, each time a reflective vehicle moves we need to reconstruct a local EM around it using surrounding REMs. Instead of a simple blending[2] between pre-computed EMs, which would provide many artifacts[3], we reconstruct a local EM by extracting and blending the correct sub-parts of surrounding REMs (See section 4.1 and 4.2). In section 4.3, we adapt our algorithm to possibly invalid REMs and we introduce an adaptative reconstruction to avoid discontinuity (See section 4.4) and control the frame rate (See section 5).

[2]The reflection direction is used to access the pixel color in the REM and a blend is applied to get the final color.

[3]For example, when a camera tracks an object, the surrounding environment reflection seems to be translating. With a simple blend, the environment would simply appear and disappear with a ghosting blur effect as you can see in figure 9 (a).

To summarize our contributions, we provide a framework to use reference environment maps, placed around a complex scene. We propose a method for computing dynamic environment maps for multiple moving objects, blending information provided by a subset of the reference environment maps; the blending is geometry-based and uses discontinuities detection, by efficiently analyzing changes in visibility in the geometry, to provide accurate reflections of the environment. With this new technique we also improve the frame rate using different levels of detail for EMs.

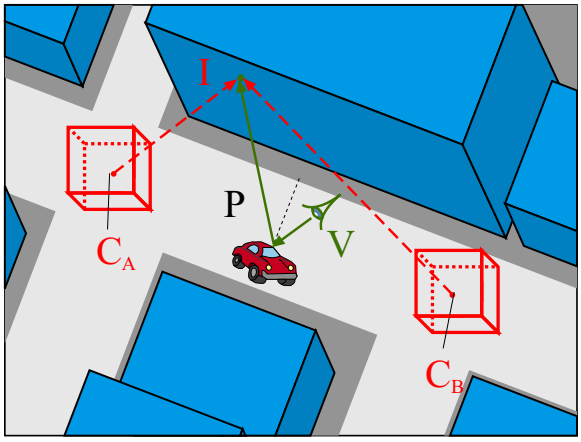


Figure 2: At the geometric point P the camera views the reflection of the scene as in point I. This information can be associated to the two pre-computed REMs (A and B). The vector C_AI can be used as texture coordinates for the EM A, and similarly the vector C_BI can be used as texture coordinates for the EM B.

4. INTERACTIVE RECONSTRUCTION OF THE LOCAL ENVIRONMENT MAP

4.1 General Idea

This section explains the heart of our method. Given a point P on the reflective object, we need to deduce a reflected color from the pre-computed REMs. We use here the term *point* in the general sense of geometric point, we will see next paragraph how this idea is used in practice. As describe on figure 2, the normal of the object in P and the view-point position V define the reflected ray by through the angle of reflection about the normal being equal to the angle of incidence, with the incident and reflected ray in the same plane. This reflected ray intersects the surrounding environment at a point I. With the classical local illumination model, the color at the point P is the sum of the ambient/diffuse term of the object with the color of I (specular term). If we assume that reflected objects such as building are only diffuse, the color of I can be associated with the surrounding two REMs A and B. Notice, we use only two REMs because this number is appropriate in our case of vehicles moving in paths defined by streets. Nevertheless, for other types of scene, one might consider a larger number of REMs if necessary.

The reflection is applied as a texture on each polygon of the reflective object combining these two REMs with multitexturing. For each vertex, a computation as previously

explains for the point P is done: a point I is computed and the vector $\overrightarrow{C_A I}$ is passed to the hardware as texture coordinates (C_A is the center of the EM A). Similarly, with the multi-texturing, the vector $\overrightarrow{C_B I}$ is used with the EM B. We avoid complex computation for each pixel of the polygon, since texture coordinates are interpolated during the rasterisation. Because the texture interpolation does not match necessary with the geometry of the environment, it may result some discontinuities. We propose a solution to this problem in section 4.4.

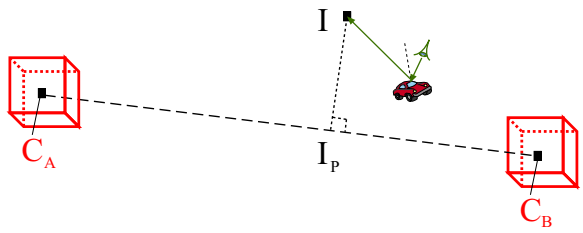


Figure 3: For each vertex, a weight for each surrounding REM is applied to take into account the distance and the validity of the computed coordinates from each REM.

For each vertex, a weight for each surrounding REM is applied to take into account the distance and the validity of the computed coordinates from each REM. The point I_p is the projection of I on the axis defined by the center of each of the two REMs. From the distance of the center of the EMs to the projected point I_p we can deduce the weight to apply on each reference map for the blending:

$$D_A = dist(C_A, I_p)/dist(C_A, C_B)$$

and

$$D_B = dist(I_p, C_B)/dist(C_A, C_B)$$

The weights to apply for the blending are $W_A = 1 - D_A$ and $W_B = 1 - D_B$, with D_A and D_B clamped between 0 to 1 as I_p can be situated outside the segment $[C_A, C_B]$. Note that $W_A + W_B = 1$.

4.2 Computation of the Intersection Point I

The technique presented in section 4.1 does not need to know which primitive of the scene is intersected by the reflective ray, only the intersection point I is needed. To find I, we avoid a complete ray tracing computation by tracing our ray from P in a discrete representation of the geometry. Notice that it could be possible to perform this ray tracing on images of the REMs since we have also the depth values, as presented in [18]. We excluded these methods because they are dependent on the number of pre-computed images, since the ray tracing has to be done in each image. Not so far from this idea, we prefer to use a heightfield representation since urban scenes are often 2.5D. Note that for other scene a regular grid could be used. The heightfield is a greyscale image of the city taken from a top view with an orthographic projection, where each pixel refers to the height of the scene. An example of the heightfield used for one of our model is presented in figure 4.

To find I, we trace the ray from P through the heightfield representation as described in [21, 2]. The ray stops when it arrives in a pixel where the value is higher than the height of the ray. We assume that the heightfield representation is close enough to the geometric representation of the scene to use the entering point I_e of the ray into the cell instead of I. By using this heightfield representation instead of the whole geometry for the ray tracing we introduce artifacts, since it does not match exactly with the geometric representation. Typically, this results in errors as in figure 9 (c) where discontinuities are not always well defined. The important point to notice is that the error introduced here is constant and continue, since we use the heightfield representation as a continuous structure. This continuity permits to avoid sudden variation on the reconstructed EM. However, there is a trade off on the resolution of the discreet representation. The more precise the resolution of the heightfield is, the more accurate the results will be. However, in terms of computation time a small resolution is preferable.

Figure 4: To find the reflected point (See figure 2) we perform a discrete ray tracing in a heightfiel representation of the scene.

4.3 Reference Environment Map Validity

As illustrated in figure 5, sometime the point I is not visible from one of the REMs which is then considered as invalid for the vertex currently considered. To test the local validity of the REM, we add a depth-test after finding the point I.

Together with the textures of the REMs, we store depth maps for each face of the cube. Then, we compare the depth value stored in the pixel corresponding to the direction $[C_A, I]$ of the depth map with the depth value of I (distance between C_A and I). If the depth value of the direction $[C_A, I]$ is less than the distance between C_A and I, this means that the REM is invalid for the vertex currently considered, because an object occludes the point I from C_A. Therefore, we cannot use this REM and associated weight W_A is set to 0. Since we compute the intersection point I for each vertex of a polygon, each vertex has its weight that can be potentially null. Notice, the depth-test is similar to the one carried out in the shadow map algorithm [26].

If all the REMs are invalid we do not have the information we are looking for, and thus a special treatment needs to be done to avoid black regions that can appear in the reflection. One solution would be to add a REM at this point. The detection of such a case could be used to add automatically a new REM at this position. In our implementation, the closest REM (the one with the highest weight before the validity test) is considered to avoid gaps regions being displayed.

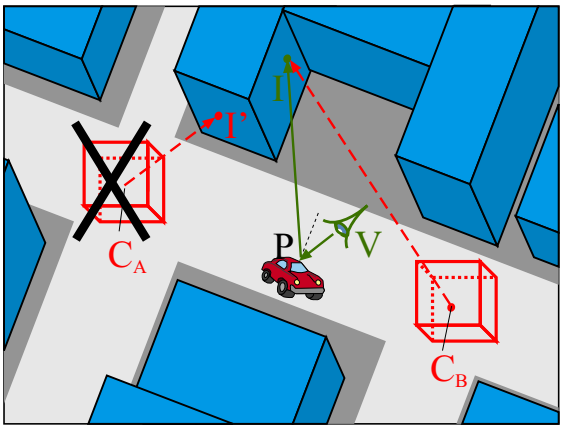


Figure 5: Example of an invalid case when using the reference maps. Point I should be reflected on the dynamic object in point P. However, it is occluded from one of the REMs, in this case A, by an object. To avoid errors in the reflection, A needs to be invalidated and dismissed when computing the local EM around the dynamic object.

4.4 Adaptative Mesh

The complete process of extracting information from the REMs could be performed at each polygon of the reflective object. However, this has several drawbacks. First, the computation cost depends on the number of polygons. Second, the distribution of the vertices might be irregular, resulting in distortion and discontinuity of the reflection for some areas and too high precision for others. Third, because reflections depend on the camera position, if the object stops moving, we still have to perform our algorithm when the camera moves.

For this reasons, we decided to compute first a local EM as the one presented in figure 6, and then use it as a regular EM. Thus, we are able to control the operation cost and the desired quality. The reconstruction is done separately for each of the six images of the cube with an off-screen rendering. The aim is to reconstruct faster an EM without rendering the whole scene, but with a similar quality. Such a reconstructed EM is presented figure 7.

We compute each face of the new EM with a mesh on which we apply the process described in the previous sections 4.1, 4.2 and 4.3. We start with a regular mesh 10×10 that will be refined if necessary. For each vertex of this mesh, we trace a ray starting from the center of the moving object and going through the vertex. Then the intersecting point is used to compute the texture coordinate of the vertex and two weights for the two surrounding REM are computed.

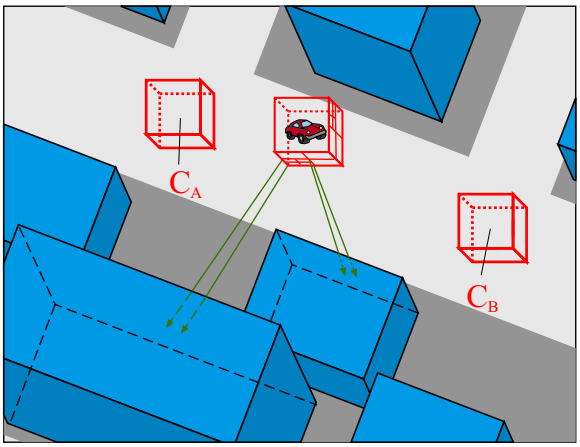


Figure 6: Illustration of the decision on the mesh subdivision of the EM, depending on the depth variation at vertices.

The reflected environment has some depth discontinuities to which our algorithm is sensitive, resulting in some popping effects. The method used to overcome this problem is illustrated in figure 6. We recursively subdivide the initial mesh depending on the depth variations of the reflected point at the vertices. Given a quad of the mesh described by the vertices $1, 2, 3, 4$, and the associate depth values Z_1, Z_2, Z_3, Z_4, we subdivide if our criteria is true. Our criteria is true if one of $|Z_i - Z_j|/min(Z_i, Z_j) > R$ is true for $i, j = 1, 2, 3, 4$, $i \neq j$ and where R is a constant ratio chosen to reflect important changing in the scale of our model. This criteria means that if there is a large variation in depth between vertices of a patch of the mesh, the patch is subdivided until the appropriate level is found. The maximum subdivision corresponds to either a threshold set by the user or to the size of a pixel. A result of this subdivision process is shown in figure 7.

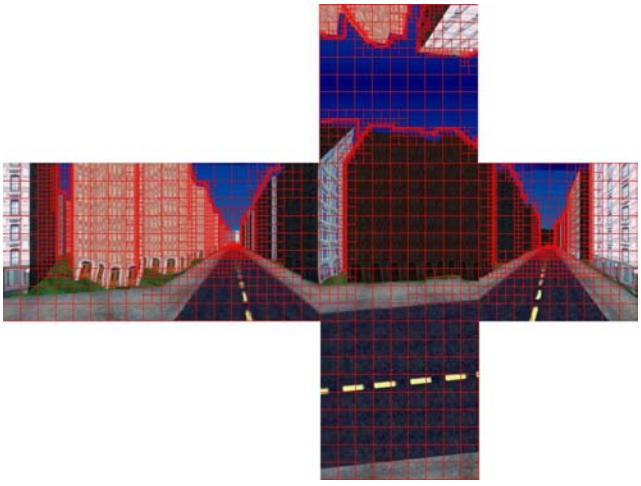


Figure 7: An example of a reconstructed EM. In red, one can see the adaptive mesh subdivision depending on the depth variation at vertices.

5. ADAPTIVE QUALITY FOR CONTROLLING THE INTERACTIVITY

In our case, we need to consider multiple objects in very complex environments. Therefore, we absolutely need to be able to control and adapt the rendering cost to keep a constant frame rate. For example, the algorithm can be modified to treat the reflections differently, depending on the viewpoint distance or on the degree of interest of the user. With our algorithm we can produce several levels of detail by controlling the resolution of the mesh of the reconstructed EM. Two other parameters, also existing in the case of re-rendering all the geometry, can be added: controlling the resolution of the reconstructed images of the EM and deciding which blending to apply for far objects.

We control the resolution of the mesh of the reconstructed EM according to the size of the object on the screen. The size of the smallest quad of our adaptive mesh (See figure 7) does not have to be smaller than a pixel after being projected on the final image. For a dynamic object, we then compute the size in pixels of the projected bounding box of the dynamic object on the screen. We divide this size by the resolution of the EM (128 or 256 pixels in our implementation) to obtain the minimum size in pixels of the quads that compose our mesh. Notice, this number can be deliberately increased to give a blur effect to the reflection that can fake motion blur and then significantly increases the realism of the dynamic object.

When objects are sufficiently far away, a simple blending is applied. As shown in section 7, this also significantly helps to accelerate the results.

6. IMPLEMENTATION AND DISCUSSION

Our simulation runs on a 2Ghz Intel Pentium PC with a GeForce4 Ti4600 graphics card. The basic display is handled by Performer^{TM} on Linux, and therefore we benefit from view-frustum culling and the scene graph. We also use the hardware capacity to accelerate the rendering. Our method, consist in computing the texture coordinates of each vertex with a ray tracing on the heightfield, which is done in software. Then the hardware capacity is used for the environment map rendering. Since we do not use unusual features of graphics hardware, our implementation will be able to run both on SGI and on PCs that could drive different kinds of display: CAVE-like, immersive desk, etc.

Moreover, according to the latest nVIDIA information on ray tracing in the vertex engine[4], we believe that the performance could be improved with the latest graphics boards (GeForce FX). Indeed, this generation of board will allow enough functionality to compute the texture coordinates (mostly the ray tracing in heightfield) in a vertex shader program.

Our method provides an efficient solution to reconstruct EMs around moving objects. Due to the pre-computation of the REM, our technique is able to manage only static objects like buildings as object that can be reflected. Of course the reflecting objects are dynamic. It is clear that static objects are the major components of a typical scene. Nevertheless, we can render other dynamic objects on the reconstructed EM to reflect both static and dynamic geometry. We have started to explore this direction by rendering dynamic virtual humans [30] in a second pass on the reconstructed EM (See images (e), (f) and (g) on figure 10). We have not implemented yet the rendering of other dynamic vehicles on the reconstructed EM because of technical difficulties about Performer.

[4]Ray-traced Reflection Demo, http://www.cgshaders.org/shaders

7. RESULTS

Since the complete reconstruction of the "Place Massena" in Nice has not been achieved yet, we have tested our technique in a virtual city. This virtual model provides extended polygonal complexity and thus dynamic objects can provide complex reflective results. In this paper, we focused on the rendering of reflection effects, and therefore the behavior of objects, especially for cars, inside the virtual scene is very simplistic.

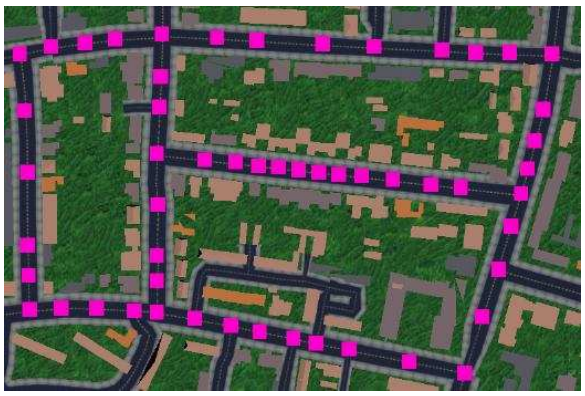
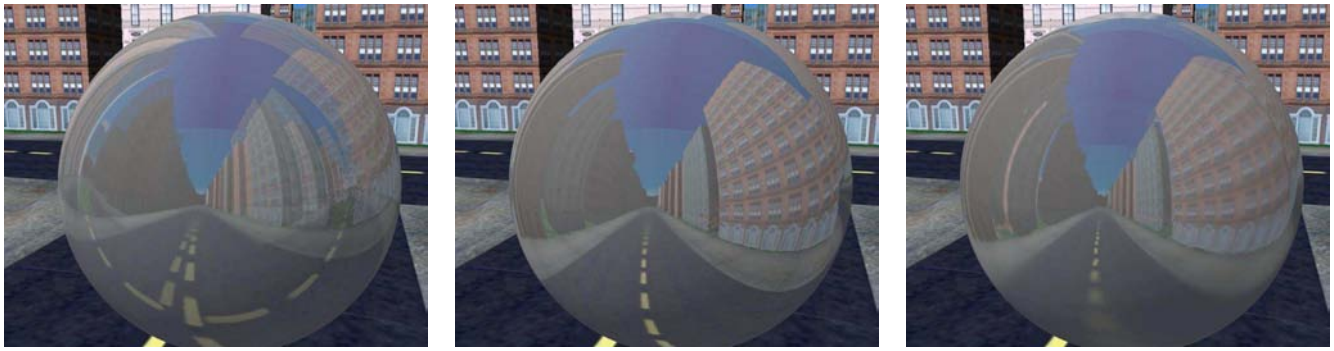


Figure 8: Cropped view of the distribution of the reference environment maps.

We applied our algorithm for the case of vehicles moving around a virtual city composed of buildings (25,600 triangles) and 8 trees (59,100 triangles each) with a total of $\tilde{5}$00,000 triangles. The local model where the simulation runs has 8 roads and covers an area of 1 km^2. For our heightfield ray tracing the size of one cell is 1 meter. The simulation we tested uses 54 REMs placed as shown in figure 8. Each EM has a resolution of 128x128 pixels for each of its 6 faces, with 4 bytes per pixel (3 bytes for the color plus one for the depth), using 6x65Kb=390Kb of memory. The total memory used to store all the reference maps is therefore 54x390=21.6Mb without using any compression. Texture compression gives at least a 4:1 compression ratio while introducing very limited artifacts (compression ratio can go up to 6:1 with nVIDIA cards). If we assume a compression ratio of 4:1, we consume 5Mb of texture memory. With a graphics board of 128Mb of memory we can apply our technique in a city 10 to 15 times bigger.

We measured the scalability of the algorithm in relation to the number of dynamic objects, as it is summarized in the table shown in table 1. In a first experiment (column *No reflection*), we perform the simulation using the OpenGL Phong rendering: diffuse and specular ($N.L$) without any reflection. We compare the technique consisting of re-render



(a) (b) (c)

Figure 9: (a) Using a simple blending between REM produces ghost effects in the blend. (b) Rendering the entire scene on the 6 faces of the EM gives a good image quality but it is difficult to achieve that in real time for several objects. (c) Using geometric information and REM, our algorithm provides accurate reflections.

# of cars	# of cars displayed	No reflection (Fps)	Without LOD		With LOD	
			Re-rendering EM (Fps)	MEM (Fps)	Re-rendering EM (Fps)	MEM (Fps)
15	3-5	14	0.2-0.8	4	1.2	8
30	10	8.5	0.1	1.3	1	5-6

Table 1: Table showing the different frame rates obtained for a scene with moving vehicles. We compare our algorithm to others using just Phong shading, or with the complete re-computation of environment maps for each moving object.

the whole scene onto the 6 faces of the cube (column *Re-rendering*) with our technique (column *MEM*) in both conditions with or without levels of detail.

Using our method increases greatly the visual quality and therefore the realism. For certain cases, it might be that the error introduced due to approximation makes the result of the blend blurred compared to the computation of the environment map by rendering the entire scene. In figure 9, we compare the result of the reflection of the environment on a sphere accessing reflection information from a surrounding environment EM. In (a), the EM results from a simple blending between two REMs. In (b), we computed the EM from scratch, rendering the whole scene onto each face of the cube. In (c), we applied our MEM technique, interpolating between two REMs using an adaptive mesh on the cube and taking into account the discontinuities of the environment. Image (a) is obviously completely wrong. Image (b) has a very high quality. The results of our method, as shown in image (c), are very close to image (b), but the reflection is less sharp. However the obtained reflection is very similar to the one obtained with environment maps, thus validating our method. Also the case of the reflection on the sphere is one of the worst since the object has a regular distribution of the normals. For an object as complex as a car, the imprecisions would not be perceptible.

In term of relative performance, our technique is much more efficient comparing to the re-rendering of the scene on a new EM, with a factor from 6 to 8 times faster when levels of detail is applied. A comparison between the column *No reflection* and the column *With LOD* shows that our technique is only from 1.5 to 1.75 time slower than rendering vehicles without any reflection, while our technique increases greatly the realism. In term of absolute performance, one thing to consider is that the frame-rate is already low for 30 cars simulated without computing the reflections (See column *No reflection*). This is due to the large number of polygons (500 000) of the scene. Another thing to consider is that Performer^{TM}performs only view-frustum culling, and therefore cars behind the buildings also use the multiple environment maps technique. Using occlusion culling and point-base rendering for trees could significantly improve the frame rate.

In figure 10, the quality of the results is illustrated for different configurations, showing that the algorithm works for a wide range of geometry and not only for simple objects constraint on the floor. Small objects like trees are the bottleneck of our method, producing easily inaccurate reflections, since the precision of the reflection is strongly dependent on the size of the heightfield representation used for the ray tracing. Nevertheless, quality of the results for this kind of objects are reasonable as you can see figure 7 (a).

8. CONCLUSION AND FUTURE WORK

In this paper, we presented a technique that makes use of the properties of environment maps to enable the reflections of a complex environment onto dynamic objects. We presented a new method, called multiple environment maps, that allows the use of a set of pre-computed environment maps, placed around the scene, that dynamically deduce new environment maps associated with the dynamic objects. We thus avoid rendering the scene six times per object at every frame. To reconstruct the environment maps, we merge reference environment maps using geometric information to retrieve the

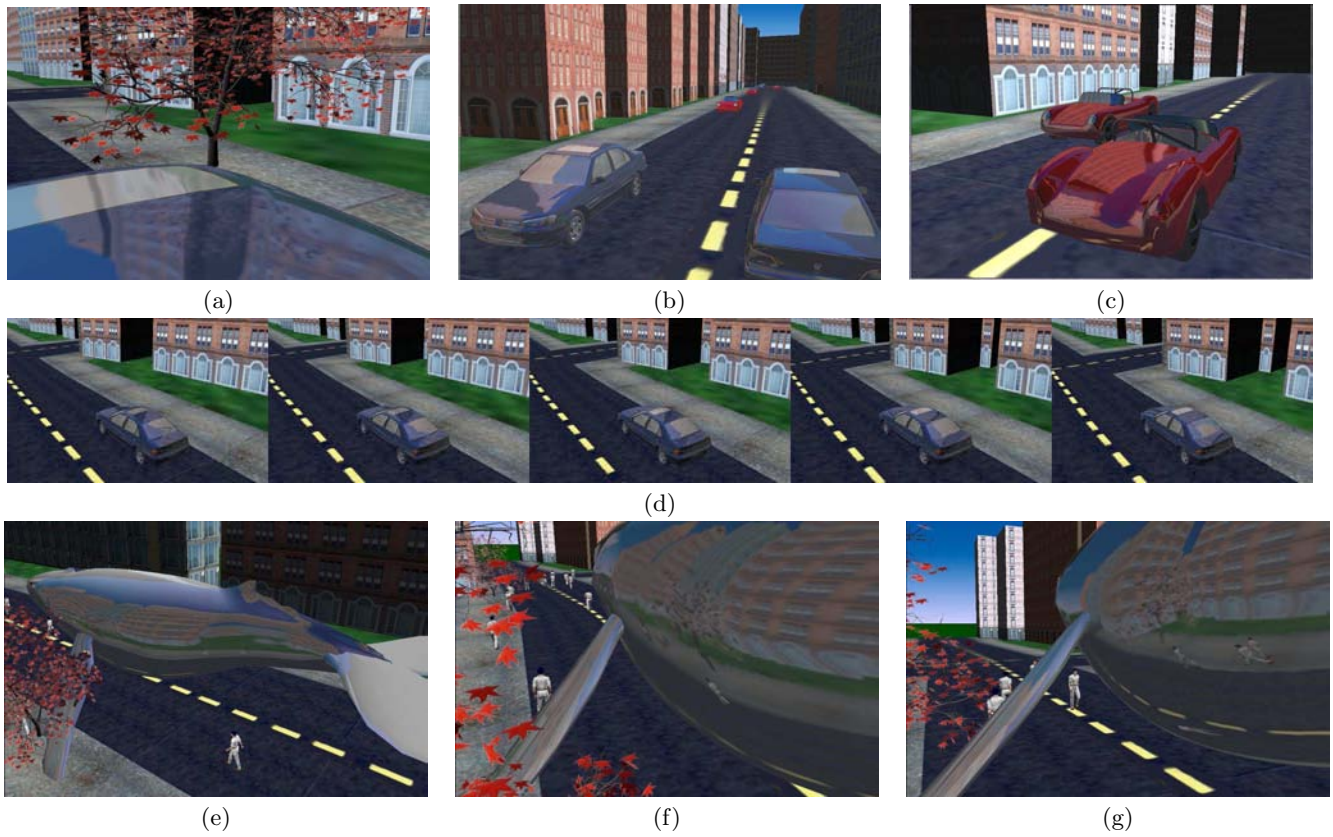


Figure 10: (a), (b), (c) Close views on reflective moving cars in a city. (d) Sequence of animation of a virtual car driving in a virtual city. (e), (f), (g) Invasion of the city by a whale, reflective trees and animated virtual humans.

correct texture coordinates. We control the quality and the complexity by sampling the faces of the EMs with a resolution adapted to the geometry. Moreover, we applied levels of detail on the environment maps to accelerate the rendering. The visual quality of the results and the computation times are satisfactory, and they prove that combined with other acceleration techniques, such as occlusion culling or levels of detail of the geometry, the technique can be extremely powerful and used for virtual reality applications.

However, this method needs to be more automatic. First, REMs should be automatically positioned in the environment. Second, the development of the levels of detail technique with new metrics could allow EMs to be reused for several objects or several frames. Third, this method could be generalized for any environment (not just cities). Instead of using a heightfield, a 3D grid could store the information for allowing any kind of static geometry like arch, balcony, etc. It would, also, be interesting to reconstruct our EM around dynamic objects by extracting informations from more than two REMs since common graphics hardware allow up to eight textures simultaneously. Finally, work on the behavior of vehicles and specially on the interaction between vehicles and crowd has to be done to increase the global realism of the simulation.

Acknowledgement

This work has been funded by the European Union though CREATE (contract number IST-2001-34231) which is a 3-year RTD project funded by the 5th Framework Information Society Technologies (IST) Programme of the European Union. We wish to thank Mel Slater and Hila Ritter Widenfeld for proofreading this paper.

9. REFERENCES

[1] Daniel G. Aliaga and Anselmo Lastra. Automatic image placement to provide a guaranteed frame rate. *SIGGRAPH 1999, Computer Graphics Proceedings*, pages 307–316, 1999.

[2] J. Amanatides and A. Woo. A fast voxel traversal algorithm for ray tracing. In *Eurographics '87*, pages 3–10, August 1987.

[3] ATI. Car paint. *http://mirror.ati.com/technology/wp/carpaint.html*, 2002.

[4] J. F. Blinn and M. E. Newell. Texture and reflection in computer generated images. *Communications of the ACM*, pages 542–547, October 1976.

[5] Chris Buehler, Michael Bosse, Leonard McMillan, Steven J. Gortler, and Michael F. Cohen. Unstructured lumigraph rendering. In *SIGGRAPH 2001, Computer Graphics Proceedings*, pages 425–432, 2001.

[6] Brian Cabral, Marc Olano, and Philip Nemec. Reflection space image based rendering. In *Siggraph 1999, Computer Graphics Proceedings*, pages 165–170, 1999.

[7] Eric Chen and Lance Williams. View interpolation for image synthesis. In *SIGGRAPH 1993, Computer Graphics Proceedings*, pages 279–288, August 1993.

[8] Shenchang Eric Chen. Quicktime VR - an image-based approach to virtual environment navigation. In *SIGGRAPH 1995, Computer Graphics Proceedings*, pages 29–38. ACM SIGGRAPH, August 1995.

[9] Paul E. Debevec, Camillo J. Taylor, and Jitendra Malik. Modeling and rendering architecture from photographs: A hybrid geometry- and image-based approach. In *SIGGRAPH 1996, Computer Graphics Proceedings*, pages 11–20, August 1996.

[10] Shachar Fleishman, Daniel Cohen-Or, and Dani Lischinski. Automatic camera placement for image-based modeling. *Computer Graphics Forum*, June 2000.

[11] S.J. Gortler, R. Grzeszczuk, R. Szeliski, and M.F. Cohen. The lumigraph. In *SIGGRAPH 1996, Computer Graphics Proceedings*, pages 43–54, August 1996.

[12] Ned Greene. Environment mapping and other applications of world projections. *IEEE Computer Graphics and Applications*, 6(11):21–29, 1986.

[13] Ziyad Hakura, John M. Snyder, and Jerome E. Lengyel. Parameterized environment maps. In *2001 ACM Symposium on Interactive 3D Graphics*, pages 203–208, March 2001.

[14] Ziyad S. Hakura and John M. Snyder. Realistic reflections and refractions on graphics hardware with hybrid rendering and layered environment maps. In *Rendering Techniques 2001: 12th Eurographics Workshop on Rendering*, pages 289–300, June 2001.

[15] W. Heidrich and H.-P. Seidel. Realistic, hardware-accelerated shading and lighting. In *SIGGRAPH 1999, Computer Graphics Proceedings*, pages 171–178, 1999.

[16] Jan Kautz and Michael D. McCool. Approximation of glossy reflection with prefiltered environment maps. In *Proceedings of Graphics Interface 2000*, pages 119–126, 2000.

[17] M. Levoy and P. Hanrahan. Light field rendering. In *SIGGRAPH 1996, Computer Graphics Proceedings*, pages 31–42, August 1996.

[18] Dani Lischinski and Ari Rappoport. Image-based rendering for non-diffuse synthetic scenes. In *Rendering Techniques '98*, pages 301–314, 1998.

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

[20] G.S. Miller and C.R. Hoffman. Illumination and reflection maps: Simulated objects in simulated and real environments. In *SIGGRAPH '84 Advanced Computer Graphics Animation seminar notes*. July 1984.

[21] F. Kenton Musgrave. Grid tracing: Fast ray tracing for height fields. Technical Report YALEU/DCS/RR-639, Yale University Dept. of Computer Science Research, 1988.

[22] NVidia. Cube environment mapping. *http://developer.nvidia.com/view.asp?IO=Cube_Mapping_Paper*, 2000.

[23] University of Saarbruecken. Open rt. *http://www.openrt.de/Publications/index.html*.

[24] Paul Rademacher, Jed Lengyel, Edward Cutrell, and Turner Whitted. *Measuring the Perception of Visual Realism in Images*. Springer Wien, New York, NY, 2001.

[25] RealViz. Imagemodeler. *http://www.realviz.com*.

[26] W. T. Reeves, D. H. Salesin, and R. L. Cook. Rendering antialiased shadows with depth maps. In *SIGGRAPH 1987, Computer Graphics Proceedings*, pages 283–291, July 1987.

[27] Steven M. Seitz and Charles R. Dyer. View morphing: Synthesizing 3D metamorphoses using image transforms. In *SIGGRAPH 96 Conference Proceedings*, pages 21–30, August 1996.

[28] J. Shade, S.J. Gortler, L. He, and R. Szeliski. Layered depth images. In *SIGGRAPH 1998, Computer Graphics Proceedings*, pages 231–242, July 1998.

[29] Wolfgang Stuerzlinger. Imaging all visible surfaces. In *Proceedings of Graphics Interface 99*, pages 115–122, 1999.

[30] Franco Tecchia, Céline Loscos, and Yiorgos Chrysanthou. Imagebased crowd rendering. *IEEE computer graphics and applications*, 22(2):36–43, March/April 2002.

[31] Jay Torborg and Jim Kajiya. Talisman: Commodity Real-time 3D graphics for the PC. In *SIGGRAPH 96 Conference Proceedings*, pages 353–364, August 1996.

[32] Douglas Voorhies and Jim Foran. Reflection vector shading hardware. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 163–166. ACM Press, 1994.

[33] T. Whitted. An improved illumination model for shaded display. *Communications of the ACM*, pages 343–349, 1980.