REAL-TIME RECURSIVE SPECULAR REFLECTIONS ON PLANAR AND CURVED SURFACES USING GRAPHICS HARDWARE

Kasper Høy Nielsen **Niels Jørgen Christensen**

Informatics and Mathematical Modelling
The Technical University of Denmark
DK 2800 Lyngby, Denmark
{khn,njc}@imm.dtu.dk

ABSTRACT

Real-time rendering of recursive reflections have previously been done using different techniques. However, a fast unified approach for capturing recursive reflections on both planar and curved surfaces, as well as glossy reflections and interreflections between such primitives, have not been described. This paper describes a framework for efficient simulation of recursive specular reflections in scenes containing both planar and curved surfaces. We describe and compare two methods that utilize texture mapping and environment mapping, while having reasonable memory requirements. The methods are texture-based to allow for the simulation of glossy reflections using image-filtering. We show that the methods can render recursive reflections in static and dynamic scenes in real-time on current consumer graphics hardware. The methods make it possible to obtain a realism close to ray traced images at interactive frame rates.

Keywords: Real-time rendering, recursive specular reflections, texture, environment mapping.

1 INTRODUCTION

Mirror-like reflections are known from ray tracing, where shiny objects seem to reflect the surrounding environment. Rays are continuously traced through the scene by following reflected rays recursively until a certain depth is reached, thereby capturing both reflections and interreflections. Although common in ray traced images, the realistic shading of highly reflective surfaces is seldomly seen in real-time virtual reality applications. However, pipeline rendering features available in consumer graphics hardware, such as texture mapping and environment mapping, can be used to approximate some of these effects.

Environment mapping methods are normally used to achieve effects that resemble ray tracing on curved surfaces (e.g. a sphere), by modelling reflections as distant. Standard environment mapping methods are only a function of direction and therefore ignore motion parallax and self reflection, i.e. environment mapping only takes the reflection direction into account and ignores the position of the reflected ray. This can be noticeable for reflections of objects close to the reflector, since every point on an environment mapped surface sees the exact same surroundings regardless of its position. Static precalculated environment maps can be used to create the notion of reflection on shiny objects. However, for modelling reflections similar to ray tracing, reflections should capture the actual synthetic environment as accurately as possible, while still being feasible for interactive rendering. In dynamic scenes, reflections need to be updated to match the actual state of the scene.

Although environment mapping techniques can be used to render reflections on planar surfaces (e.g. a floor), this is normally not used in practice, since it can lead to severe artifacts. As an environment map captures the surrounding environment with respect to a single fixed viewpoint, an environment mapped planar reflector will miss important motion parallax effects. Instead planar reflections are traditionally captured by rendering reflected geometry from a mirror viewpoint.

The recursive concept in ray tracing can generally be adapted to pipeline rendering in a simplified form. Recursive reflections allow reflective objects to appear in other reflections. Instead of achieving this ef-

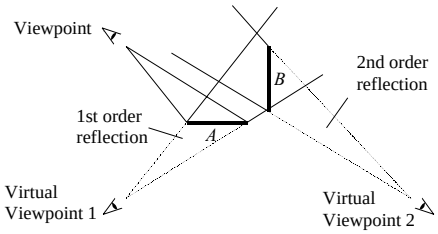

Figure 1: Second order recursive planar reflection: The reflected view generated by reflector A sees another reflector B and processes this reflection in a recursive manner.

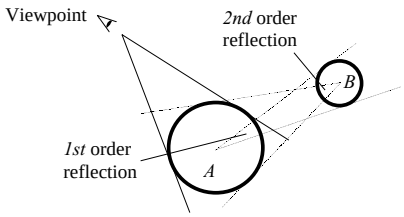

Figure 2: Second order curved mirror reflection in a scene with two reflective spheres. Object A contains object B in its reflection. By recursion, object B also contains object A in its reflection.

fect by tracing individual rays recursively through a scene as in ray tracing, reflective objects can be rendered explicitly into reflection images of other reflectors. By extending existing reflection methods, this means rendering other planar or curved reflections into previously generated reflection images in a recursive manner. See Fig. 1 and Fig. 2. We will refer to this concept as *reflection tracing* [Niels00]. As in ray tracing, the maximum recursion depth can be determined to limit the maximum number of interreflections to be rendered.

Previous work have performed hardware rendering of recursive reflections using different techniques, yet have not described an approach for capturing recursive reflections on both planar and curved surfaces, as well as interreflections between such primitives. This paper addresses this topic. The goal is to design a method that:

1. Efficiently captures reflections and interreflections on both planar and curved surfaces.
2. Uses texture-based techniques in order to approximate glossy reflections by low-pass filtering reflection images.
3. Works in both static and dynamic environments.
4. Uses rendering features available in low-end consumer graphics hardware.
5. Integrates easily into virtual reality applications.

To achieve our goal, we design a novel recursive texture-based method for capturing recursive planar reflections. We then extend this method to also capture recursive curved reflections, by using two different existing environment mapping techniques. We describe and compare these two closely related methods that meet our goals. In the next section, we discuss previous work and their limitations, and conduct a detailed discussion of existing methods related to our approach. Section 3 describes our approach, and section 4 discusses the results. Finally, section 5 draws the conclusion and points out directions of future work.

2 PREVIOUS WORK

2.1 Overview

The use sphere, cube and dual paraboloid mapping for rendering mirror reflections on curved surfaces (both convex and concave) is thoroughly described in the literature [Green86, Voorh94, Heidr98, Heidr99, Kilga99a, Kilga99b, McRey00, Watt00]. A good discussion of their pros and cons can be found in [McRey00]. A visual comparison of environment mapping versus ray tracing can be found in [Watt00]. Other methods have been proposed for rendering reflections: Warped *Geometric Reflections* [Ofek98], *Tetrahedron Environment Mapping* [Forte00], and *Extended Environment Mapping* [Cho00]. [Ofek98] and [Cho00] can capture recursive reflections. However, unlike sphere, cube and dual paraboloid environment mapping, these methods are not supported by graphics hardware today, but could well be candidates for future hardware implementations. While methods have been developed for capturing recursive reflections on curved surfaces using sphere [Kilga99a] and cube mapping [McRey00], none of these methods consider planar surfaces. Planar recursive reflections are captured using either the stencil-buffer [Diefe97, Kilga99c, McRey00] or texture mapping [McRey00]. An interesting alternative is suggested in [Basto98], but it cannot run on todays graphics hardware due to the use of forward mapping. An approximate technique that uses ray-sphere intersections to calculate environment map lookups is also described in [Gritz01]. Finally, the use of reflection mapping techniques to approximate more general BRDF-based reflection models have been addressed by several authors [Mille84, Green86, McRey00, Cabra99, Heidr99, Basto99, Wynn00, Kautz00]. In the next two subsections we conduct a detailed discussion of existing methods related to our approach.

2.2 Planar Reflections

The methods for capturing recursive reflections on planar reflectors [Diefe97, McRey00] are generally based on beam tracing [Heckb84]. The entire view

frustrum of a planar reflector is bent and the mirrored view is followed in a recursive manner (see Fig. 1). Given a scene with n reflective surfaces and a recursion depth of d, the method has a time complexity of $O(n^{d+1})$. However, as pointed out by Diefenbach [Diefe97], culling can greatly reduce the actual number of n for each iteration.

Using multi-pass rendering, reflections are implemented by first rendering the scene without the mirrored surfaces. A second rendering is performed for each reflected viewpoint, and the resulting image is applied to the reflector. This process can be repeated recursively in scenes with multiple reflectors. Stencil operations are arranged so the reflected images are masked by the stencil buffer.

McReynolds et al. [McRey00] also describe how texture maps can be used to store the final reflection images. The stencil buffer is used for rendering interreflections into the reflection textures (reflection maps), and texture mapping is used for rendering the final image. [McRey00] state that one of the advantages of the texture mapping technique compared to the stencil technique, is that it may be acceptable to use reflection textures with the contents from the previous frame in a dynamic environment.

2.3 Curved Reflections

Recursive reflections on curved surfaces can be approximated using environment mapping techniques. By pre-generating environment maps for reflected objects, these can be used for rendering interreflections into other environment maps [Kilga99a, McRey00] (see Fig. 2). The regeneration of many environment maps can however be computationally expensive in a dynamic environment, especially when using complex environment map representations (e.g. cube maps). This needs to be solved in order to make our algorithm efficient.

One solution is to use a simpler representation, such as a (view-dependent) sphere map. A sphere map should normally be generated by warping a view-independent environment map, yet approximate techniques can also be used. One technique is to only generate a single distorted projection of the reflection as seen from the reflecting object in the direction of the virtual viewer, and use this as an approximate sphere map [Palli99]. The projection can be generated by using a high field-of-view and/or post-warping the rendered view using non-linear distortion. This technique does not produce an accurate environment map since the back and parts of the sides of the environment are disregarded, and since the generated image does not accurately model a sphere. Still, the technique can produce acceptable results.

An alternative solution, described by [McRey00], approximates interreflections by taking advantage of view-independent environment mapping and frame-to-frame coherence. The method can be described as: When rendering a frame, we generate top-level (i.e. recursion level 0) environment maps for each reflecting object in the scene, one at a time. During generation of an environment map E, we determine the set of reflecting objects that will be visible in this environment map. For each object we check:

1. If an environment map has already been generated for this object in the current frame, we use this environment map for rendering the object into environment map E.
2. If such an environment map does not exist, we check if an environment map has been generated for this object in the previous frame. If so, we use this instead for rendering the object into environment map E.
3. If no environment map has been generated we draw the interreflecting object into E without reflection. Note, that this is only the case the first time an object appears, e.g. in the first frame.

The method uses previously generated environment maps to approximate reflections of interreflecting objects. At frame i the method captures between $(i * n)$ and $(i * n + n - 1)$ interreflections, where n is the number of reflectors[1]. Thus, the method does not make use of recursion. In dynamic environments the method regenerates n environment maps for each frame, and therefore has a time-complexity of $O(n)$. In static environments the method runs in constant time, after initializing the environment maps with the desired number of interreflections. As the method relies on previously generated environment maps for rendering interreflections, there will be a small error in the rendered interreflections compared to real recursive interreflections, when using the method in a dynamically changing environment. The error is largest for the first environment map rendered, as all interreflections here are based on previous frames, and smallest for the last environment map, as all reflections here are based on environment maps generated in the current frame. The error depends on the frame rate and the speed of dynamic changes. However, in general it can be hard to distinguish flaws in secondary reflections on curved objects. Furthermore, the higher the frame rate is on a real-time system, the smoother the animation is, and the smaller the error will be between successive interreflections. Thus, there is a good chance that such errors hardly will be noticeable on a system exhibiting smooth, fluid animation.

[1] With the exception of $n = 1$, where the number of interreflections equals 0, regardless of i.

3 OUR APPROACH

By using beam-tracing principles it is possible to capture specular interreflections. Yet, the method is restricted to planar surfaces, and cannot account for interreflections on curved surfaces. Methods exist for capturing approximate interreflections on curved surfaces using environment mapping. However, none of these methods consider planar surfaces.

While beam-tracing can be computationally expensive, this concept appears to be the only accurate way to simulate recursive planar reflections using graphics hardware. McReynolds et al. [McRey00] have shown how to capture planar interreflections using texture mapping. Yet, as interreflections are captured using the stencil-buffer, the method does not allow for glossy interreflections. Furthermore, because such textures need to be of relatively high resolution to capture reflections accurately, the method can be memory intensive as a reflection map is generated for each reflecting object before rendering.

We propose an alternative planar method that does allow for glossy interreflections and which has constant low memory requirements that depend on the selected recursion depth. We extend this method to also capture curved reflections and interreflections using environment mapping. We design two variations of our algorithm: One, based on a view-dependent environment representation, and one based on a view-independent representation. Both have advantages and disadvantages which we discuss in section 5. For simplicity we first describe our planar method (section 3.1), and then extend the method to curved surfaces (section 3.2).

3.1 Planar Reflections

The concept of our planar reflection tracing method is simple: For each rendering of the scene we generate a reflection map for each reflecting surface, and apply it onto the reflector during rendering. As each reflection map in itself is a rendered view of the scene from a mirror viewpoint, this process can be repeated recursively until a certain depth d is reached.

Because of the limited amount of texture memory, it is generally wasteful to compute the reflections *before* rendering. Instead we propose that reflections are computed *during* rendering, thereby minimizing memory requirements. Using this scheme, only one reflection map is required for each recursion level, as illustrated in Fig. 3. Note, that the method implicitly requires render-to-texture capabilities as the back-buffer constantly is occupied by the contents of the current frame. When a reflecting surface is met, during rendering of the polygons in a scene, the rendering task is immediately discontinued and the corresponding reflection map is generated. After generation, control is returned to the rendering task, the reflection map is applied to the reflector, and the rendering task is continued. Reflection maps are thus generated, used, and discarded, and the texture space can therefore be reused for subsequent rendering of reflections. The pseudo code for the algorithm is shown in Fig. 4. The time complexity for this algorithm is $O(n * (n-k)^d)$, where n is the total number of objects, and k is the number of non-reflecting objects. The method can therefore be computationally expensive, particularly for deep recursion levels, e.g. $d \geq 3$. However, as in [Diefe97], the actual runtime of the method can be improved by using culling techniques. Moreover, to further reduce scene complexity, it can be advantageous to use low level-of-detail objects (LODs) when rendering interreflections. The resolution of the generated reflection maps can also be dynamically scaled, based on the projected area of the reflector, thus minimizing rendering overhead for

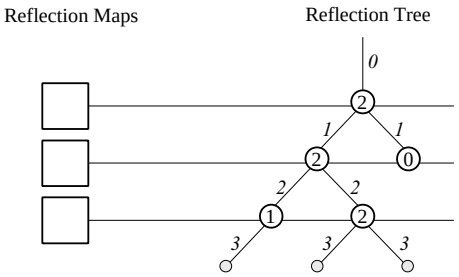

Figure 3: Example reflection tree, showing the assignment of one reflection map for each recursion level. The number of reflections encountered at each node is shown circled, while the recursion level is shown in italics.

```
DrawScene(view, depth) {
  Clear frame buffer
  for (each object i in scene) {
    if ((object is reflector) &&
        (depth < maxdepth)) {
      Select refl.map n = depth as
        render-target
      Setup mirror_view
      DrawScene(mirror_view, depth + 1)
      Restore render-target
      Apply refl.map n to object i
    }
    Draw object i
  }
}
```

Figure 4: Pseudo code for rendering a scene with recursive planar reflections. *DrawScene* renders the scene from a previously defined *view*. When a reflecting object is met, DrawScene is called recursively. The recursion *depth* is passed as a parameter and used to determine the render-texture level n, and termination of the reflection tracing branch, based on *maxdepth*. An initial call of *DrawScene(view, 0)* is used for rendering the entire scene.

distant reflections. Finally, as in ray tracing, good results can be achieved with low recursion depths (e.g. $d < 3$), depending on the specularity of surfaces. For surfaces exhibiting subtle specular reflections, reflection contributions are diminished significantly for each recursion level, and it may be acceptable to terminate a reflection tracing branch if the specular level goes below a certain threshold.

We note that the described scheme precludes any use of reflection coherence between successive frames, as the reflection maps are discarded after use. We choose to trade speed for memory, because planar reflections are only rotation-invariant and must be regenerated for new viewpoints, causing the practical use of coherence to be somewhat limited.

3.2 Planar and Curved Reflections

The described reflection tracing process can be extended to also handle curved reflections, by generating environment maps for curved reflectors instead of planar reflection maps. This can be achieved by generating view-dependent environment maps, e.g. sphere maps through warped cube maps, or by using approximate sphere mapping techniques.

For generating an environment map for a reflecting object, that object is normally hidden to avoid self-reflection. Yet, for letting each environment map contain interreflections of both planar and curved surfaces, the environment map must be generated by recursion. As the hidden object should be visible at deeper recursion levels, it is necessary to account for the hidden objects during the recursion process. With this in mind, the planar algorithm can be extended to also handle curved objects by using the pseudo code shown in Fig. 5.

Still, since some environment mapping methods are view-independent, it seems reasonable to exploit this during the reflection tracing process. Unlike planar reflection maps, view-independent environment maps can draw reflections seen from different viewpoints and can therefore be reused for generating different views of the same reflector.

It is advantageous to use a view-independent parameterization for top-level reflections, as these can be re-used for successive frames in momentarily static scenes. For interreflecting objects, the view-independence means that an environment map only needs to be generated once per frame, and that a reflecting object that sees itself through another reflecting object, in theory can use the same environment map to generate the interreflection. However, this poses a problem: For generating an environment map for an object that sees itself through another reflecting object, we may actually attempt to use the environment map that we are currently working on. One solution is to first render all the non-reflecting geometry into the environment map, thereby ensuring that the environment map can be used as a reflection, but it will still miss one of more interreflections. Another approach is to take advantage of temporal coherence by using previously generated environment maps to approximate higher order reflections. The method described by McReynolds et al. [McRey00] is advantageous, since it has a linear time-complexity, while still capturing deep levels of interreflections in both static and dynamic environments. We extend the proposed planar algorithm to include curved surfaces by using this method. This requires generation of view-independent environment maps for each curved reflector, before rendering the main frame (recursion level 0). Since recursive calls might access the environment map currently being generated, a complete representation of the environment map should be available at all times. To achieve this, an environment map should first be updated when rendering of the entire map is completed, i.e. similar to a double buffering scheme. Using this technique, the combined algorithm for capturing both curved and planar reflections is as shown in Fig. 6. As the planar recursion is dominant, the time-complexity becomes $O(n*(n-k-c)^d)$, where n is the total number of objects, k is the number of non-reflective planar objects, and c is the number of curved objects.

```
DrawScene(view, depth, objref) {
  Clear frame buffer
  for (each object i in scene) {
    if ((object is reflector) &&
        (depth < maxdepth)) {
      if (objref) Unhide object objref
      Select refl.map n = depth as
        render-target
      if (curved object) {
        Hide object i
        Setup env_view
        DrawScene(env_view, depth + 1, i)
        Unhide object i
      } else {  // planar
        Setup mirror_view
        DrawScene(mirror_view,depth + 1,NULL)
      }
      Restore render-target
      Apply refl.map n to object i
      if (objref) Hide object objref
    }
    Draw object i
  }
}
```

Figure 5: Pseudo code for rendering a scene containing recursive planar and curved reflections, where a single approximate view of the environment is used as a view-dependent environment map representation for curved reflectors. Again, the recursion *depth* is passed as parameter. An extra *objref* parameter ensures that an object, hidden during environment map generation, is made visible during recursive rendering. An initial call of *DrawScene(view, 0, NULL)* is used for rendering the entire scene.

```
DrawScene(view, depth, objref) {
  Clear frame buffer
  // create view-indep. environment maps
  if (depth == 0) {
    for (each reflecting curved object i) {
      Hide object i
      Assign unique cube map to object i
      for (each cube face j) {
        Select cube face j as render-target
        Setup env_view for cube face j
        DrawScene(env_view, depth + 1, i)
        Restore render-target
      }
      Unhide object i
      Update cube map
    }
  }
  // draw the objects
  for (each object i in scene) {
    if ((object is planar reflector) &&
        (depth < maxdepth)) {
      if (objref) Unhide object objref
      Select refl.map n = depth as
        render-target
      Setup mirror_view
      DrawScene(mirror_view, depth + 1, NULL)
      Restore render-target
      Apply refl.map n to object i
      if (objref) Hide object objref
    } else if (object is curved reflector) {
      Apply assigned cube map to object i
    }
    Draw object i
  }
}
```

Figure 6: Pseudo code for rendering a scene containing both planar and curved reflectors, where cube maps are used as environment map representations for curved reflectors. An environment map is generated for each curved reflector before rendering the actual scene (at depth 0). Again, *depth* and *objref* are passed as a parameters. An initial call of *DrawScene(view, 0, NULL)* is used for rendering the entire scene.

4 RESULTS AND DISCUSSION

We have implemented both variations of our algorithm: A view-dependent method (Fig. 5) and a view-independent method (Fig. 6). Both have been used to render reflections in scenes containing different numbers of planar and curved reflectors, at fixed recursion depths. The results are shown in Fig. 7, and the measured performance statistics are shown in Table 1.

All tests were conducted on an Intel Pentium III 450 MHz PC, with 128 MB RAM, running Windows, and equipped with a GeForce 3 graphics accelerator. The test program was written in C++ using DirectX 8.0. The view-dependent method used an approximate sphere-mapping technique, while the view-independent method used cube mapping. Planar reflection maps and sphere maps were drawn into texture maps with a 256×256 texel resolution and a 32 bit pixel representation. The cube map resolution was set to 128×128. For a fair comparison, our implementation did not use LODs or any culling besides backface culling. Instead, we have used simple test-scenes. However, the use of simple frustrum culling improves the frame rates by a factor of 2 to 3.

Scene	(a)	(b)	(c)	(d)
Recursion depth	2	2	2	3
Planar objects	6	1	2	3
Curved objects	1	3	2	1
Triangles in scene	1078	1106	3890	1002
View dependent method:				
Generated refl.	38	16	16	46
Fps, static/dyn.	19 fps	32 fps	29 fps	13 fps
Used texture mem.	0.5MB	0.5MB	0.5MB	0.8MB
- Traditional	1.8MB	1.0MB	1.0MB	1.0MB
View independent method:				
Gen. planar refl.	62 (26)	19 (1)	28 (4)	69 (15)
Gen. curved refl.	6*1	6*3	6*2	6*1
Fps, dynamic	10 fps	17 fps	8 fps	7 fps
Fps, static	21 fps	97 fps	44 fps	27 fps
Used texture mem.	0.9MB	1.6MB	1.3MB	1.1MB
- Traditional	1.9MB	1.4MB	1.3MB	1.1MB

Table 1: Measured performance statistics for each method run for each of the tested scenes. The table shows the number of generated reflections, the measured frame rates, and the amounts of used texture memory. The number of planar reflections generated in static scenes is written in parenthesis. For comparison, the memory used when using a traditional method is also shown.

We observe that both planar and curved recursive reflections can be captured using the proposed methods. The results show that the methods can be computationally expensive for deep recursion levels, due to the recursive nature of the algorithms. However, this was expected as the algorithms are based on beam-tracing principles. Still, real-time frame rates have been achieved for all of the tested scenes. Although the methods are based on texture-based reflection techniques, they exhibit reasonable memory requirements. In the view-dependent implementation, the requirements solely depend on d, due to the utilization of render-to-texture functionality. In the view-independent implementation, the memory requirements depend on d for planar objects, and the number of curved objects n, as a view-independent environment map is allocated for each curved object. Thus, in most cases both methods show a reduction in memory requirements compared to methods where reflections are pre-generated for each object before rendering into the frame buffer. The savings are most significant in scenes with many reflectors. The view-dependent method is the least memory intensive.

By comparing the visual quality and performance of the two methods in practice, we observe that they both have advantages and disadvantages. The view-dependent method is computationally cheap in scenes with few reflections at low recursion depths, as only one view is generated for each reflector. However, because the method requires regeneration of environment maps for viewpoint changes, it suffers from the same computational cost in both static and dynamic environments, making it difficult to capture

deep interreflections in real-time. Furthermore, the use of approximate sphere mapping results in inaccurate curved reflections.

The view-independent method is more expensive in dynamic scenes, due to the updates of a complex environment map. Yet, the method is very computationally cheap in static scenes, and appears to capture infinite recursive reflections on curved objects. The approximation of interreflections, achieved by recycling environment maps, yields good results with fluid, smooth, dynamic animation (high frame rates), but bad results for fast or jerky animation (low frame rates). Given a less complex environment map representation than cube mapping (e.g. dual-paraboloid mapping), the view-independent method would make it possible to capture deep recursive reflections faster than the view-dependent method. The method runs in linear time with respect to the number of curved reflectors in dynamic scenes, and in constant time in static scenes. Because the method efficiently exploits both view-independence and coherence, captures deep interreflections in both static and dynamic scenes, and has potential for being faster than the view-dependent method, it is in our opinion the most advantageous of the two.

In general, the environment mapping approximation works well for reflectors that are small compared to the surrounding environment, but it can be very inaccurate when a reflector's curvature is large or when reflected objects are very close to the reflector. This is a problem that our method cannot solve. To do so, we need hardware support for more sophisticated environment mapping methods, e.g. [Ofek98, Cho00]. Furthermore, the use of texture-based techniques can cause texture related magnification artifacts, resulting in a blocky looking appearance when moving too close to a reflector or using low resolution textures. Similarly, minification artifacts can arise when looking at a reflector from afar or when using high resolution textures. Both artifacts can be partially reduced by scaling the resolution to the projected screen area of a reflector. However, this reduces the sampling interval and can result in temporal aliasing, and as the maximum size of textures is limited, magnification artifacts cannot be completely eliminated. Thus, for large reflectors the method produces inferior results compared to e.g. stencil buffer reflections. Still, the lack of quality should be weighted against the fact that texture reflections let us use filtering to simulate glossy reflections.

Finally, the methods use common pipeline rendering features, and combine well with existing rendering methods, e.g. bump mapping and texture mapping, and are therefore easily integrated into a virtual reality system.

5 CONCLUSION AND FUTURE WORK

We have described a texture-based approach for capturing recursive reflections on planar and curved surfaces in both static and dynamic environments. It takes advantage of hardware texture mapping, render-to-texture and environment mapping capabilities.

We have designed two variations of our basic algorithm. Both use beam-tracing principles to capture planar reflections, but differ with respect to the parameterization used for curved surfaces: A view-dependent method, which uses a view-dependent environment map representation, generated recursively as in the planar case. And a view-independent method, which uses a view-independent environment map representation, pre-generated before each frame and used for rendering all reflections of a single object, including interreflections in successive frames. Due to exploitation of render-to-texture functionality, both algorithms reuse reflection maps when generating recursive reflections, thereby reducing memory requirements. Finally, because the methods are texture-based, glossy interreflections can be simulated by filtering reflection images.

The results show that the algorithms can be expensive for deep recursion levels. For only capturing low levels of recursive reflections, the view-dependent algorithm is found to be fastest. However, we find that the view-independent algorithm is the most advantageous of the two, since it captures deep levels of recursive reflections, runs in constant time in static scenes (for curved surfaces only), and will be faster than the view-dependent method in dynamic scenes, when given a simpler environment map representation that allows for fast regeneration.

Future work should examine extensions of the proposed methods and test their behavior in more complex scenes. It would be interesting to examine other environment mapping parameterizations for curved surfaces, e.g. dual paraboloid mapping or a more accurate view-dependent parameterization. One could also investigate ways to use different (preferably view-independent) parameterizations for planar surfaces, which would yield a faster algorithm. For faster rendering, different LODs and culling schemes should also be tested. Finally, it should be straightforward to extend the methods to also capture (approximate) recursive refraction, as well as simulate more sophisticated BRDF based reflections.

6 ACKNOWLEDGEMENTS

The authors would like to thank Tomas Möller for his encouragement and helpful comments. Thanks to Scott Cutler and Chris Wynn for additional help. This work was supported in part by the STVF project DMM and the Nordunit2 project NETGL.

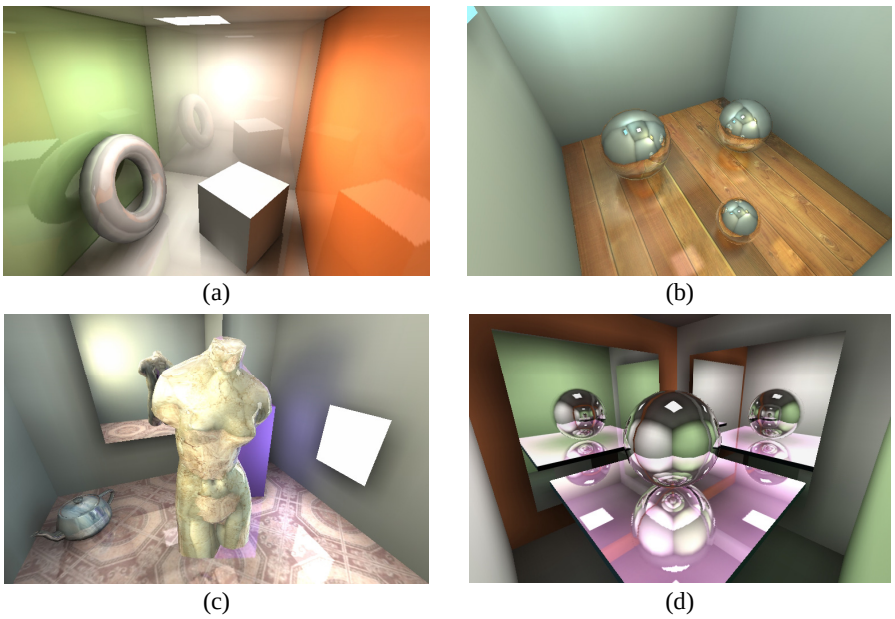

Figure 7: Test scenes containing both planar and curved reflectors. Here, scene (a) is shown rendered using the view-dependent method, while scenes (b,c,d) are rendered using the view-independent method. Scene (b) simulates a glossy floor. The static illumination in these scenes (shadows, etc.) is pre-calculated using radiosity.

REFERENCES

[Basto98] Bastos, R., Stürzlinger, W.: Forward Mapped Planar Mirror Reflections. Univ. of North Carolina at Chapel Hill, Computer Science Technical Report TR98-026, 1998.

[Basto99] Bastos, R., Hoff, K., Wynn, W., Lastra, A.: Increased Photorealism for Interactive Architectural Walkthroughs. Symposium on Interactive 3D Graphics, pp. 183-190, 1999.

[Cabra99] Cabral, B., Olano, M, Nemec, P.: Reflection Space Image Based Rendering. Computer Graphics (SIGGRAPH'99 Proceedings), 1999.

[Cho00] Cho, F. S.: Rendering Reflective and Refractive Objects with Extended Environment Mapping. Computer Science Division, Berkeley, 2000.

[Diefe97] Diefenbach, P. J., Badler, N.: Multi-Pass Pipeline Rendering: Realism For Dynamic Environments. Symposium on Interactive 3D Graphics, Proceedings, 1997.

[Forte00] Fortes, T.: Tetrahedron Environment Maps, Master's Thesis, Department of Computing Science, Chalmers University of Technology, Gothenburg, Sweden, 2000.

[Green86] Greene, N.: Environment Mapping and Other Applications of World Projections, IEEE Computer Graphics and Applications, pp. 21-29, 1986.

[Gritz01] Gritz, L., Apodaca, T., Pharr, M., Hery, C. Bjorke, K., Treweek, L. : Advanced RenderMan 3: Render Harder, SIGGRAPH 2001 Course 48, 2001.

[Heckb84] Heckbert, P. S., Hanrahan, P.: Beam Tracing Polygonal Objects, Computer Graphics (SIGGRAPH '84 Proceedings), 1984.

[Heidr98] Heidrich, W., Seidel, H. P.: View-independent Environment Maps. SIGGRAPH, Workshop on Graphics Hardware, pp. 39-44, 1998.

[Heidr99] Heidrich, W., Seidel, H. P.: Realistic, hardware-accelerated shading and lighting. Computer Graphics (SIGGRAPH '99 Proceedings), 1999.

[Kautz00] Kautz, J., McCool, M. D.: Approximation of Glossy Reflection with Prefiltered Environment Maps. Graphics Interface 2000, pp. 119-126, 2000.

[Kilga99a] Kilgard, M. J.: Real-time Environment Reflections with OpenGL, Slides, nVidia Corp.,1999.

[Kilga99b] Kilgard, M. J.: Perfect Reflections and Specular Lighting Effects With Cube Environment Mapping, Technical Brief, nVidia Corp., 1999.

[Kilga99c] Kilgard, M. J.: Improving Shadows and Reflections via the Stencil Buffer, Technical Report, nVidia Corp., 1999.

[McRey00] McReynolds, T., Blythe, D., Grantham, B., Nelson, S.: Advanced graphics programming techniques using OpenGL. SIGGRAPH 2000 Course 32, 2000.

[Mille84] Miller, G. S., Hoffman, C. R.: Illumination and Reflection Maps: Simulated Objects in Simulated and Real Environments. Course Notes for Advanced Computer Graphics Animation, SIGGRAPH'84, 1984.

[Niels00] Nielsen, K. H.: Real-Time Hardware-Based Photorealistic Rendering. Master's Thesis, Informatics and Mathematical Modelling. Technical University of Denmark, 2000.

[Ofek98] Ofek, E., Rappoport, A.: Interactive reflections on curved objects. Computer Graphics (SIGGRAPH '98 Proceedings), pp. 333–342, 1998.

[Palli99] Pallister, K.: Rendering to Texture Surfaces Using DirectX7, Gamasutra, 1999.

[Voorh94] Voorhies, D., Foran, J.: Reflection vector shading hardware. Computer Graphics (SIGGRAPH '94 Proceedings), pp. 163–166, 1994.

[Watt00] Watt, A.: *3D Computer Graphics, Addison-Wesley*, 3rd edition, 2000.

[Wynn00] Wynn, C.: Real-Time BRDF-based Lighting using Cube-Maps, nVidia Corp., 2000.