6-1994

Pipeline Rendering: Interactive Refractions, Reflections and Shadows

Paul Joseph Diefenbach
University of Pennsylvania

Norman I. Badler
University of Pennsylvania, badler@seas.upenn.edu

Pipeline Rendering: Interactive Refractions, Reflections and Shadows

Abstract
A coordinated use of hardware-provided bitplanes and rendering pipelines can create ray-trace quality illumination effects in real time. We provide recursive reflections through the use of secondary viewpoints, and present a method for using a homogeneous 2D projective image mapping to extend this method for refractive surfaces. We extend the traditional use of shadow volumes to provide reflected and refracted shadows as well as reflected and refracted lighting. A shadow blending technique is demonstrated, and the shadow and lighting effects are incorporated into our recursive viewpoint paradigm. Finally, we incorporate material properties including a translucency model to provide a general framework for creating physically approximate renderings. These techniques are immediately applicable to areas such as 3D modelling, animation and interactive environments to produce more realistic images in real time.

Keywords
real time, rendering pipeline, animation

Pipeline rendering: interactive refractions, reflections and shadows

Paul J Diefenbach and Norman I Badler

A coordinated use of hardware-provided bitplanes and rendering pipelines can create ray-trace quality illumination effects in real time. We provide recursive reflections through the use of secondary viewpoints, and present a method for using a homogeneous 2D projective image mapping to extend this method for refractive surfaces. We extend the traditional use of shadow volumes to provide reflected and refracted shadows as well as reflected and refracted lighting. A shadow blending technique is demonstrated, and the shadow and lighting effects are incorporated into our recursive viewpoint paradigm. Finally, we incorporate material properties including a translucency model to provide a general framework for creating physically approximate renderings. These techniques are immediately applicable to areas such as 3D modelling, animation and interactive environments to produce more realistic images in real time.

Keywords: real time, rendering pipeline, animation

Much attention has been devoted to photo-realistic rendering techniques as ray-tracing and radiosity packages have become increasingly sophisticated. These methods provide a basic foundation of visual cues and effects to produce extremely high quality and highly accurate images at a considerable cost, namely computation time. Neither of these techniques has any widespread application in true interactive environments, such as animation creation and virtual worlds.

Many so-called interactive environments such as Virtual Building systems[1,2] rely on precomputation of static environments to form progressive radiosity solutions. Other systems dealing with lighting effects[3] rely on a series of images from a single viewpoint. All of the systems suffer from large computational overheads and unchangeable geometry. Even in incremental radiosity solutions[4], geometry changes require significant recomputation time. In addition, radiosity-based solutions inhibit the use of reflective and refractive surfaces.

Systems based on forward ray-tracing[5] are either non-interactive or else suffer from the problems inherent in the technique[6]. Only a few attempt to handle indirect illumination accurately[7]. Backward ray-tracing systems[8–10] more accurately handle caustics, but again these methods are very time-intensive and not remotely interactive. Even the fastest ray-tracing systems require static geometry to achieve their results[11].

In contrast, advanced hardware architectures such as the SGI Reality Engine™ have brought an added level of realism to interactive environments through the use of sophisticated graphic pipelines and added levels of screen buffer information. These features have enabled software developers to bring previously unavailable details such as shadows and mirrors to many interactive applications. Even the most basic graphics systems today now support some level of image masking and manipulation, common to the image-processing community for years. These hardware provisions have yet to be fully exploited, though clever programming techniques by several implementors have produced real-time shadows and mirrors[12,13].

This paper expands these techniques to include not only reflection but a technique for refractive surfaces as well. The model presented extends the current reflection techniques to provide an arbitrary level of refraction and reflection for use in 'hall-of-mirror' type environments and to provide a close approximation for refractive objects. A corrective image transform is presented to correct for perspective distortions during the image mapping of the secondary refracted image. In addition, a method for combining the previously exclusionary shadow and mirror stencilling methods is demonstrated which not only preserves shadows in all secondary images, but which also accounts for refraction and reflection of the light and shadows in the primary and secondary images as well. Finally, the use of hardware-provided features such as fog

Department of Computer Science, University of Pennsylvania, Philadelphia, PA 19104, USA
Paper received: 16 April 1994; revised: 6 June 1994

0141-9382/94/03/0173–08

and texture blending is shown to provide simulation of varying material properties such as translucency and shininess. Combined, these techniques provide a real-time alternative to ray-tracing for creating fast, approximate reflective and refractive lighting effects. Furthermore, the techniques described provide a foundation for more advanced rendering features such as anisotropic reflections and caustics.

DEFINITIONS

For the purposes of this paper, we shall introduce terms common to users in the GL environment. *Stencil planes* are essentially an enhanced Z-buffer mentioned in Reference 14. In its simplest form, pixels are written only if the current stencil value (analogous to the current Z value) of the destination pixel passes the defined stencil test. Depending on the result, the pixel is written and the stencil value is changed.

Shadow volumes are volumes bounded by silhouette faces. A silhouette face is a face created for each edge of an object by extending that edge away from the light source along the light-ray direction.

An *accumulation buffer* is a secondary image buffer to which the current image can be added. The resulting image can also be divided by a constant. This enables a blending of images or image features.

In–out refractions are refractions that occur when light passes from one medium to another and back to the first, such as light traversing through a piece of glass. There is an entry refraction and an exit refraction, producing a refracted ray parallel to the incident ray.

REFLECTIONS

Reflections are a useful tool in interactive modelling and an important element for creating realistic animations. A reflective image corresponds to an inverted image from a secondary viewpoint. In other words, the reflected image is the flipped image from a viewpoint on the 'other' side of the mirror. This analogy provides the basis for mirror reflection in systems such as that described in Reference 13.

Mirrors are implemented by rendering the entire environment, exclusive of the mirrored surface. The mirrored surface is drawn with Z-buffering, creating a stencil mask on pixels where the mirror is visible. A second rendering of the environment is then performed from the reflected viewpoint, drawing only over the previously masked pixels. Because the reflected angle (angle from mirror plane to reflected viewpoint) is the negative of the incident angle and because the image is flipped, the reflected image directly 'fits' onto the mirror.

REFRACTIONS

Just as reflections provide strong visual cues, refractive elements also add additional realism for animations and interactive environments. Refractive images are similar in concept to reflections, but more complex in practice.

While a mirrored image directly corresponds to the reflective surface to which it maps, a refracted image maps to a distorted image space. Simply performing a second rendering in the stencilled area does not overlay the correct image portion. This is demonstrated in *Figure 1*. The area visible through the transparent surface in the refracted view is different from the image area from the original viewpoint; areas outside the refracting surface and even in front may be visible in the refracted image (*Plate 1*). This difference is due to two factors: the difference between incident and refracted viewpoints, and the perspective distortion.

Because the incident angle does not equal the refracted angle, the refracted image is rotated with respect to the original image. This is further compounded by the rotated image plane undergoing a perspective distortion different from the perspective distortion of the original plane. The perspective transformations are the same, but because the planes have different orientations, the resulting distortions are different. The result is that a refractive square planar face, for example, maps to two different quadrilaterals in the original versus the refracted images.

The refractive image I_r does correspond to the original

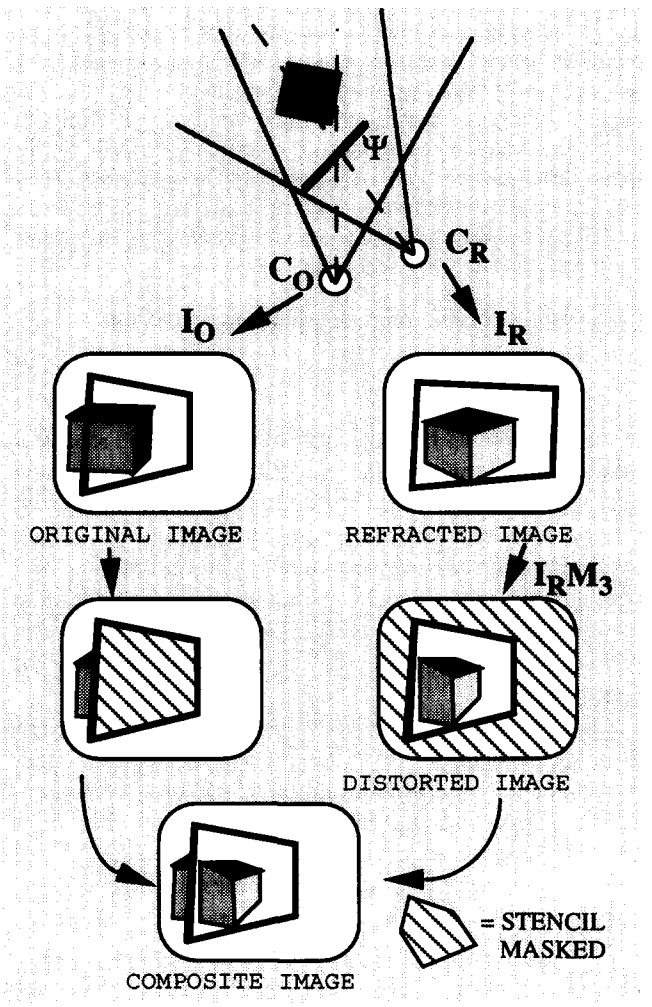


Figure 1 Refracted image versus camera image

image I_o through a 2D bijective projective mapping M_3. This mapping is the intersection of the 3D image mapping set M_4 with the reflective planar surface Ψ:

$$I_o = I_r M_3 \quad (1)$$

where

$$M_3 = M_4 \cap \Psi \quad (2)$$

and

$$M_4 = P^{-1} C_r C_o^{-1} P \quad (3)$$

In Equation (3) P is the perspective transform and C_o and C_r are the original and refracted camera transforms, respectively.

This results in a 2D projective transform of arbitrary quadrilateral to quadrilateral described in Reference 15 and included in the Appendix. This transform, described by a 3×3 homogeneous transform, can be applied directly to the screen–viewport mapping to distort the refractive image into the normal image space. In hardware that supports user-defined transforms, this transform can be inserted directly at the end of the rendering pipeline. In systems where this is not possible, such as the Silicon Graphics™ architecture, this transform can be implemented as a 4×4 homogeneous transform inserted in the world-to-unit pipeline. The resulting transform is constructed with a zero scale factor for Z so that the mapping is to the $Z = 0$ plane. Without this mapping, the tapering and skewing effects from the quadrilateral distortion affect the Z coordinates. This scaling does, however, preclude the use of the Z-buffer for hidden surface removal as all image points now have the same Z value. This method also does not allow for the translucency simulation described below, due to the loss of depth.

Note also that this method does not produce true refractions, merely a close approximation to the refractive image. In a true refractive image, every ray incident to the refractive plane bends according to its angle with the plane; this method, however, uses only one incident angle. In practice, two angles are used to provide more realistic results with the system. First, the incident ray is taken from the camera location to the refracting face centre to determine whether the incident angle is greater than the critical angle. If this is the case, the surface is taken to be wholly reflective. If the angle is less than the critical angle, the incident angle for Snell's Law is taken at the point of intersection of the view vector (camera's negative Z axis) and the plane in which the refracting face lies. This method ensures that the critical angle is reached as the plane moves tangentially to the view, yet the refracted image is seen as a smooth scrolling of the background behind the face.

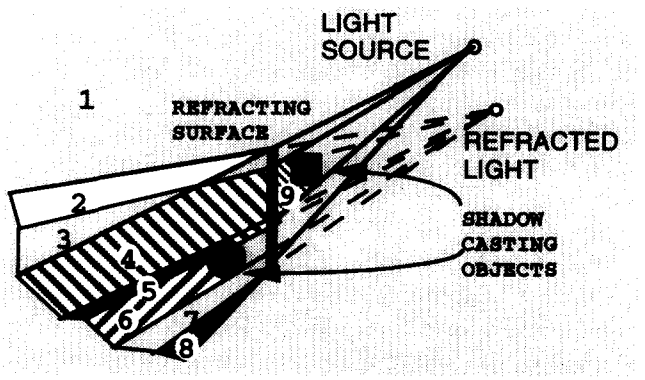
REFRACTIVE SHADOWS*

While the above method does produce accurate reflective images and close approximations for refractive surfaces, it does not produce accurate lighting effects from these surfaces. Light reflects off a mirror and refracts through glass, producing different shadows than if not present. To produce a more accurate image, these effects must also be taken into account. Therefore, any shadow-generation method must not only work in conjunction with the stencilling method described above, but it must also be affected by the reflective and refractive surfaces in a scene.

Our shadows are implemented using the traditional shadow volume technique described by Heidmann[12]. This technique uses the in–out principle of silhouette faces to mask regions inside the shadow volume.

To understand how this method must be extended for refractive surfaces, examine *Figure 2*. This figure displays the complex shadow patterns caused by objects on both sides of a refracting surface. Note that this is not an exact representation but instead a hybrid model used in our system to demonstrate the refracting effects more clearly. The rays are refracted as in a change of medium; they do not represent true in–out refraction of a material with a thickness. With in–out refraction, the refracted rays are parallel to the incident rays and merely offset, thereby not permitting direct light to fall within the light volume[16]. Although the included images were generated with this change-of-medium model, in–out refractions are achieved merely by changing the refracting function (or by placing back-to-back refracting faces with opposing indices of refraction in the current model).

To model shadows accurately, each of the above-mentioned features must be included in our shadow model. To accomplish this, we require a two-pass shadow-generation approach. The first phase generates all shadows and lighting falling within the refracted light area. The second pass renders all lighting and shadows outside this area. This method creates both the shadow and caustic effects of the refractive surface.



1: PRIMARY LIGHT
2: PRIMARY AND REFRACTED LIGHT (NOT PRESENT WHEN EXIT REFRACTIONS)
3: CUBE'S REFRACTED SHADOW & PRIMARY LIGHT
4: CUBE'S REFRACTED SHADOW
5: CUBE'S REFRACTED SHADOW & SPHERE'S SHADOW FROM REFRACTED LIGHT
6: SPHERE'S SHADOW FROM REFRACTED LIGHT
7: REFRACTED LIGHT
8: REFRACTING SURFACE'S SHADOW

Figure 2 Light interaction with refractive surface

*Although only refractions are mentioned henceforward, the methods described are applicable for reflections with only minor variations

In the first pass, a light volume[17] is generated for the refracting face. Shadow volumes are then generated for shadows falling inside this volume. This itself includes two cases, objects inside the volume generating shadows, and objects outside the volume whose shadows get refracted into the volume. In the first case, the shadow volume cannot intersect the refracting plane, for to do so would place the object outside the light volume. In the second case, the shadow volume must intersect the refracting plane in order to be refracted into the light volume. Because true in–out refraction results in refracted rays parallel to incident rays, objects outside the light volume cannot cast shadows into the light volume directly from the primary light source. Both intersection cases can be checked during the shadow volume generation. A simple pre-shadow generation check using dot-products can determine if the object is on the appropriate side of the refracting plane and can save having to generate the shadow volume.

The second pass creates shadows for the entire environment. Even the refracted light volume region is included. This captures the shadow effect caused by the refractive surface itself.

RECURSION

Both methods for rendering shadows and for rendering refraction and reflection require use of the stencil planes. While it might seem that the refraction stencil mask value would be a logical choice for the zero value in the shadow algorithm, this is not the case. In order to have recursive refractions, we instead choose a value which is three-quarters of the maximum stencil value for our 'zero' shadow value, and one-half of the maximum for our minimum shadow stencil value. This provides half of the stencil buffer for shadow calculation and half for recursive levels. These values can be adjusted according to the recursion level needed or the shadow object complexity.

We choose zero for our render area value; this is the stencil mask value for drawing at every level of recursion. At each level of recursion, all values less than the stencil minimum are incremented by one (setting the current rendering area to one), and the new refractive surface is drawn setting the stencil value to zero. The refracted image is then drawn in the zero stencil area, and the process is repeated for all other refractive surfaces. Once the desired recursion level has been reached, all stencil values less than the shadow minimum are decremented (with zero capped), which essentially puts us back one level of recursion. The process is then repeated for the next retractive surface, with stencil values incremented by one and the surface creating a stencil mask of zero. This process is illustrated in *Figure 3* and *Plate 2*.

At each level of recursion, shadows must be drawn in the valid area. The reason for our choice of shadow zero is now apparent; it avoids conflict with our recursive refraction levels. All stencil values of zero at each level are changed to the shadow zero, and shadows are then rendered as described above using the in–out method. The shadow zero should be chosen so that the in–out method does not go

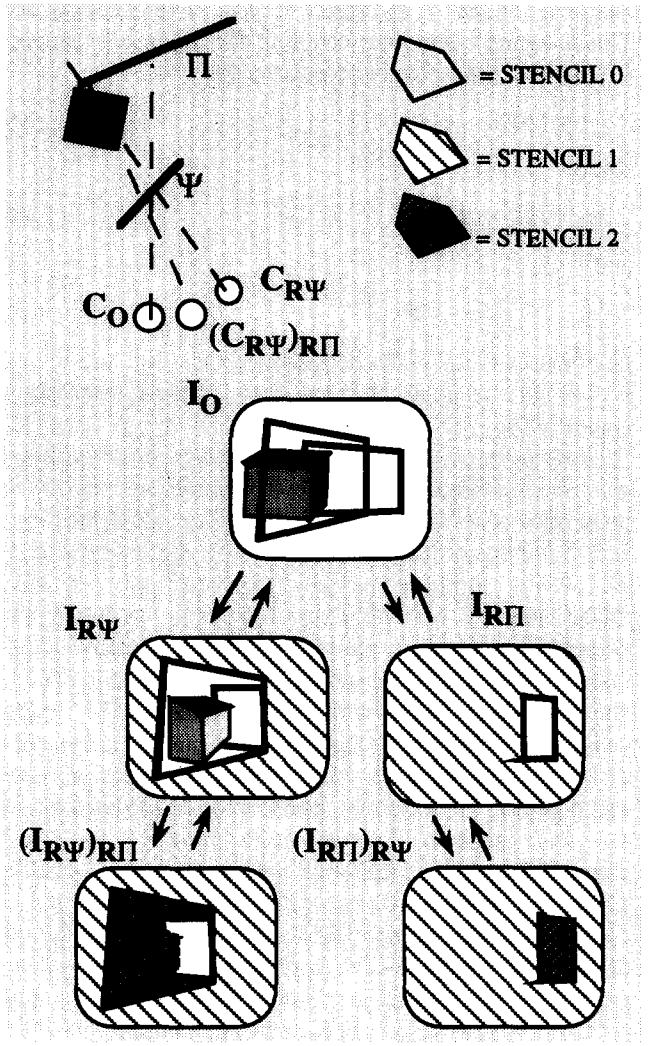


Figure 3 Recursive stencilling

below the shadow minimum or above the maximum stencil value in order to prevent conflict with the refraction stencils. After all shadows are drawn, all stencil values greater than shadow minimum are reset to zero, for continuation of the refraction recursion. The entire procedure is seen in *Figure 4*.

SOFT SHADOWS AND CAUSTICS

While the above algorithm does handle the constituent effects of primary and indirect illumination, diffuse light is incrementally added, resulting in producing only the umbra of the shadows. In order to produce the penumbra, an accumulation of the lighting effects from the primary as well as refracted lights must occur. Fortunately, we can use an accumulation buffer to do just this.

There are two methods for performing this accumulation of lighting effects. The first method is to treat each shadow calculation as independent and sum each resulting image. Areas that receive light from both the source and refracted light volume produce caustic effects (*Plate 3*). This method has limitations when using a single accumulation buffer due to the non-independent effects of different lights.

```
drawwindow(Camera)
{
  if (SHADOWS){
    turn_lights_off();
    draw_normal_objects(Camera,ZERO); //ambient only
    for (each light){
      apply_stencil(EQUAL, ZERO, REPLACE, SHAD_ZERO);
      draw_ref_shadows();    //draw shadows in ref areas
      apply_stencil(GREATER, SHAD_MIN, REPLACE, SHAD_ZERO);
      draw_all_shadows();    //draw remaining shadows
      apply_stencil(GREATER, SHAD_MIN, REPLACE, ZERO);
    }
  }else
    draw_normal_objects(Camera,ZERO);
  if (REF){
    REF_LEVEL++;
    draw_ref_objects(Camera);
    REF_LEVEL--;
  }
}

apply_stencil(COMP_FUNC, COMP_VALUE, PASS_FUNC, PASS_VALUE)
{
  stencil(COMP_FUNC, COMP_VALUE, PASS_FUNC, PASS_VALUE);
  apply_to_screen();  //apply stencil to every pixel
}

/*SHADOW ROUTINES*/

draw_ref_shadows()
{
  for (each ref_face){
    face_light_stencil(ref_face); //light volume
    turn_light_on(light);
    make_all_shadows(MUST_INTERSECT, ref_face);
    draw_normal_objects(Camera,SHAD_ZERO);
    apply_stencil(GREATER, SHAD_MIN, REPLACE, SHAD_ZERO);
    ref_light(light,ref_face);    //move to ref position
    make_all_shadows(CAN'T_INTERSECT, ref_face);
    draw_normal_objects(Camera,SHAD_ZERO);
    apply_stencil(GREATER, SHAD_MIN, REPLACE, SHAD_ZERO);
  }
}

draw_all_shadows()
{
  make_all_shadows(ALWAYS, ref_face);
  draw_normal_objects(Camera,SHAD_ZERO);
  apply_stencil(GREATER, SHAD_MIN, REPLACE, SHAD_ZERO);
}

make_all_shadows(MODE,ref_face)
{
  for (each face)
    draw_shadow_volume(face,MODE,ref_face);
}

draw_shadow_volume()
{
  make_shadow_volume(ref_face,sv,intersect_face);
  switch (MODE){
    case CAN'T_INTERSECT:
      if (!(intersect_face==ref_face)){
        draw_shadow_stencil(sv);
        make_shadow_volume(intersect_face,MODE,ref_face);
      }
    case MUST_INTERSECT:
      if (intersect_face==ref_face){
        make_shadow_volume(intersect_face,ALWAYS,0);
      }
    case ALWAYS:
      draw_shadow_stencil(sv);
      if (intersect_face){
        make_shadow_volume(intersect_face,ALWAYS,0);
      }
}

/*REF ROUTINES (Refraction and Reflection)*/

draw_ref_objects(Camera)
{
  if (REF_LEVEL==1)
    clear_stencil(ZERO);
  apply_stencil(EQUAL, ZERO, REPLACE, REF_LEVEL);
  for (each ref_face){
    stencil(EQUAL, REF_LEVEL, REPLACE, ZERO);
    draw_face(ref_face);  //REF_LEVEL->0 where face
    RefCamera=ref_camera(ref_face,Camera);
    draw_window(REF_CAMERA);
    apply_stencil(EQUAL, ZERO, REPLACE, REF_LEVEL);
  }
  apply_stencil(EQUAL, REF_LEVEL, REPLACE, ZERO);
}

draw_normal_objects(Camera,STEN_VALUE)
{
  setup_view(Camera);      //view from camera
  stencil(EQUAL, STEN_VALUE, KEEP, STEN_VALUE);
  for (each non_ref object)
    draw_object();
  }
}
```

Figure 4 Recursion procedure

The second method uses an extension of shadow volumes[14] for soft shadows. By processing all shadow volumes without producing intermediate images, the stencil value of each pixel represents a 'darkness level' due to encasement in several shadow volumes. The actual lighting of the scene is performed after all shadow volumes have been generated. Assuming the darkness level is only greater than the zero stencil value, the scene is redrawn with diffuse lighting for areas whose stencil value is less than the shadow zero value. All stencil values are then decremented, and the image is redrawn. This process is repeated for each darkness level, accumulating each intermediate image. This produces a final image with intensities based on the number of enclosed shadow volumes. This method does not suffer from the problems inherent in the first method; however, extensive stencil value 'juggling' of the image is necessary.

OBJECT ATTRIBUTES AND TRANSLUCENCY

Refraction and reflection need not be limited to purely transparent or purely specular surfaces, respectively. We can create a multitude of materials by rendering the refractive surface again after the refractive image is drawn. This second rendering is alpha-blended with the refracted scene. On the Iris Indigo™ system, this actually requires two additional renderings of the refractive face since lit faces cannot have a source alpha value. The first rendering is done without colour and sets the destination alpha value to the appropriate value. The second rendering is with lighting and a blending function depending on the destination pixel chosen. This permits hardware shading effects and other hardware rendering features such as textures to be blended with the refracted scene.

In addition, translucency can be simulated using the

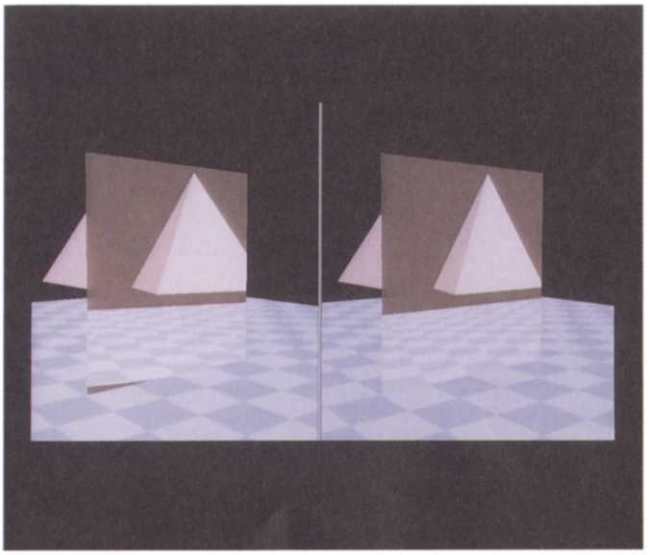


Plate 1 Undistorted and distorted refraction

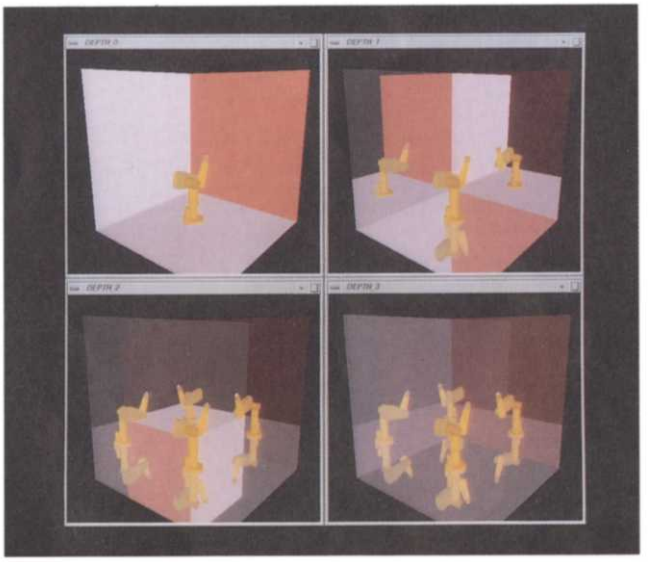


Plate 2 Recursive reflection

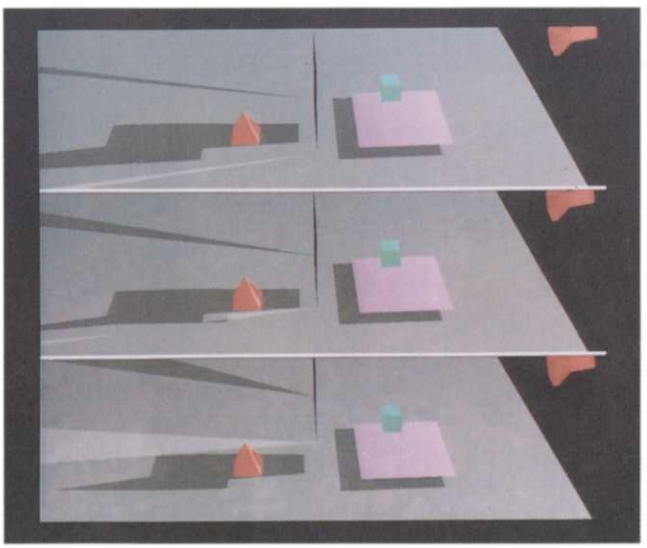


Plate 3 Refracted caustics and shadows

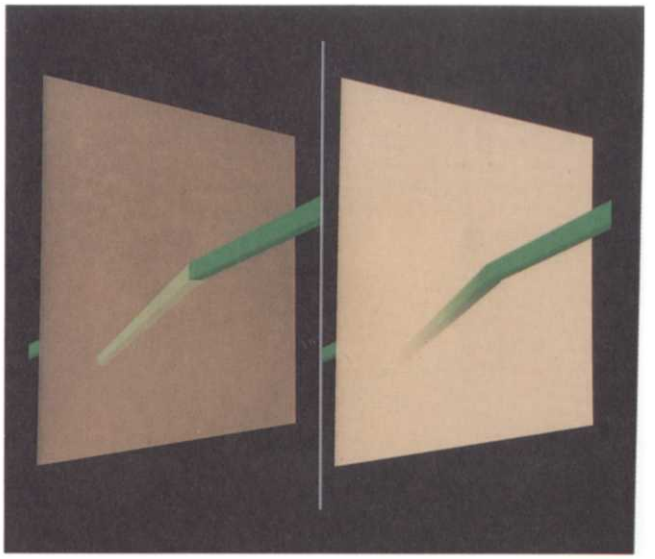


Plate 4 Effects of translucency

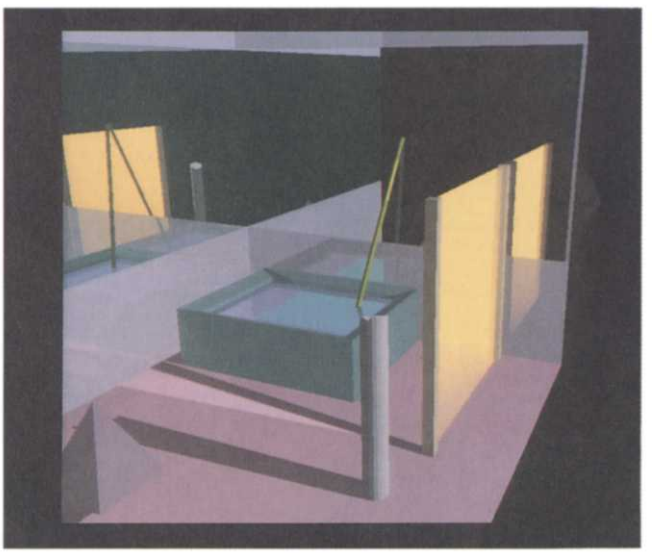


Plate 5 Combined effects

hardware fog feature. Translucent objects act as a filter, with closer objects more clearly visible than more distant objects due to the random refractions that take place. This effect can be approximated using hardware fog features with the minimum fog set at the refractive plane distance and the maximum at the desired distance depending on the material property. Although fog is linear with respect to the view, the approximation is fairly accurate due to the limited angular displacement of the refracting plane owing to the critical angle. The effect of translucency versus simple alpha blending is demonstrated in *Plate 4*. The combined effects are seen in *Plate 5*.

LIMITATIONS AND EXTENSIONS

While the system described can produce fast, complex images, it does suffer from several shortcomings. Because it relies on multiple viewpoints, refractive and reflective surfaces must be planar. Shadows suffer from aliasing effects due to the use of image space precision in the

calculation. In addition, the hardware shading (typically the Phong model[18]) used for the illumination model is widely known for its inadequacies[19]. In the same regard, the system is hardware dependent on the number of stencil and accumulation bits, as well as the viewport–screen transforms supported.

The rendering phase is also very time-dependent on the complexity of the environment as well as the recursion level for refractions (reflections). Shadows require

$$O(num\text{-}edges * recursive\text{-}depth) \quad (4)$$

for each scene rendering, of which there are

$$num\text{-}ref\text{-}faces * (num\text{-}ref\text{-}faces - 1)^{(recursive\text{-}depth-1)} \quad (5)$$

For complex refractive surfaces, this expense can quickly become prohibitive even when compared with ray-tracing.

For scenes with a limited number of planar refractive and reflective surfaces, or with a low recursive depth, this system is very effective even with minimal hardware support. The system currently runs effectively on an Iris Indigo XS24™ with 8 stencil bits and a 48-bit accumulation buffer.

Additional hardware support would provide greater facilities for creating more complex images. Multiple accumulation buffers would provide greater shadow-blending capabilities. Additional pipeline control such as viewport transforms or additional fog features would enable distorted refractions in conjunction with translucency.

The method can also be extended using the accumulation buffer to handle partially reflective and partially refractive surfaces, instead of merely switching at the critical angle. Anisotropic reflections[19] could be simulated based on the orientation of the reflecting plane. Speedup can be achieved by precomputation of shadow volumes during static geometric periods. Visibility checks between refractive surfaces can also be used to reduce the number of scene renderings.

CONCLUSIONS

We have described a series of techniques for adding realism to interactive environments and producing fast animations. These techniques have the common thread of using the hardware pipeline itself to produce illumination effects commonly found only in non-interactive renderers such as ray-tracers.

This paper is presented not only as an introduction to new methods, but to serve as a platform from which to incorporate other hardware techniques to build a complete interactive renderer, as described in Reference 20.

More control for direct manipulation of the rendering pipeline is needed as well as more complex hardware lighting models. With advanced hardware features finding exotic uses in producing effects such as texture-mapped shadows, this stresses the need for greater flexibility in user interaction.

While there will always be a need for complex, very accurate rendering packages, many situations require fast, approximate solutions. The techniques outlined in this paper provide such solutions for fast animation and interactive modelling.

REFERENCES

1 Airey, J M, Rohlf, J H and Brooks, F P 'Towards image realism with interactive update rates in complex virtual building environments'. *ACM SIGGRAPH* 1990, **24**(2), 41–50
2 Teller, S J and Séquin, C H 'Visibility preprocessing for interactive walkthroughs'. *ACM Comp. Graphics* 1991, **25**(4), 61–69
3 Dorsey, J. *PhD Thesis* Cornell University, Ithaca, NY (1993)
4 Chen, S E 'Incremental radiosity: An extension of progressive radiosity to an interactive image synthesis system'. *ACM Comp. Graphics* 1990, **24**(4), 135–144
5 Glassner, A S (ed) *An Introduction to Ray Tracing*, Academic Press, London, 1989
6 Watt, A and Watt, M *Advanced Animation and Rendering Techniques*, Addison–Wesley, Reading, MA, 1992
7 Kajiya, J 'The rendering equation'. *ACM Comp. Graphics* 1986, **20**(4), 143–150
8 Arvo, J 'Backward ray tracing, developments in ray tracing'. *SIGGRAPH Course Notes* 1986, **12**
9 Heckbert, P S and Hanrahan, P 'Beam tracing polygonal objects'. *ACM Comp. Graphics* 1984, **18**(3), 119–127
10 Chattopadhyay, S and Fujimoto, A 'Bi-directional ray tracing'. *CG International '87* (1987) pp 335–343
11 Séquin, C H and Smyrl, E K 'Parameterized ray tracing'. *ACM Comp. Graphics* 1989, **23**(4), 307–314
12 Heidmann, T 'Real shadows — real time'. *IRIS Universe* 1991, **Nov–Dec**,
13 Kingsley, E C, Schofield, N A and Case, K 'Sammie'. *ACM Comp. Graphics* 1981, **15**(3), 163–169
14 Brotman, L and Badler, N 'Generating soft shadows with a depth buffer algorithm'. *IEEE Comp. Graphics Applic.* 1984, **4**(10), 71–81
15 Heckbert, P S Master's Thesis, University of California, Berkley (1989)
16 Kay, D S and Greenberg, D 'Transparency for computer synthesized images'. *ACM Comp. Graphics* 1979, 158–164
17 Nishita, T 'A shading model for atmospheric scattering considering luminous intensity distribution of light sources'. *ACM Comp. Graphics* 1987, **19**(3), 303–310
18 Phong, Bui-Thong 'Illumination for computer-generated pictures'. *Commun. ACM* 1975, **18**(6), 311–317
19 Ward, G J 'A ray tracing solution for diffuse interreflection'. *ACM Comp. Graphics* 1988, **22**(4), 85–92
20 ten Hagen, P J W, Kujik, A A M and Trienekens, C G 'Display architecture for VLSI-based graphics workstations'. *Advances in Computer Graphics Hardware I, Record of First Eurographics Workshop on Graphics Hardware* 1986, pp 3–16

APPENDIX

2D Quadrilateral projective map

The mapping of a quadrilateral to quadrilateral can be accomplished by composing the mapping of a quadrilateral-to-square with a square-to-quadrilateral. The two mappings are adjoints of each other symbolically, and therefore only the square-to-quadrilateral mapping will be given. (This mapping is also equal to the perspective transform of the camera rotation with respect to the normal of the refracting plane, which is available directly from the matrix stack.)

$$M_{sq} = \begin{bmatrix} a & d & g \\ b & e & h \\ c & f & i \end{bmatrix}, \tag{A1}$$

where

$$g = \begin{vmatrix} \Sigma x & \delta x_2 \\ \Sigma y & \delta y_2 \end{vmatrix} / \begin{vmatrix} \delta x_1 & \delta x_2 \\ \delta y_1 & \delta y_2 \end{vmatrix}$$
$$h = \begin{vmatrix} \delta x_1 & \Sigma x \\ \delta y_1 & \Sigma y \end{vmatrix} / \begin{vmatrix} \delta x_1 & \delta x_2 \\ \delta y_1 & \delta y_2 \end{vmatrix} \tag{A2}$$

$$\begin{aligned} a &= x_1 - x_0 + gx_1 & d &= y_1 - y_0 + gy_1 \\ b &= x_3 - x_0 + hx_3 & e &= y_3 - y_0 + hy_3 \\ c &= x_0 & f &= y_0 \end{aligned} \tag{A3}$$

3D Quadrilateral projective map

Given the 2D quadrilateral projective transform

$$M_{qq_3} = \begin{bmatrix} a & d & g \\ b & e & h \\ c & f & i \end{bmatrix}, \tag{A4}$$

we create the 3D transform

$$M_{qq_4} = \begin{bmatrix} a & d & 0 & g \\ b & e & 0 & h \\ 0 & 0 & 0 & 0 \\ c & f & 0 & i \end{bmatrix}, \tag{A5}$$

which clears depth values and disables Z-buffering and fog.