PORTRAY – AN IMAGE SYNTHESIS SYSTEM

Darwyn R. Peachey

Department of Computational Science
University of Saskatchewan
Saskatoon, Canada

ABSTRACT

PORTRAY is an image synthesis system which uses ray tracing to produce realistic images of three-dimensional scenes. Scenes are described to PORTRAY in a high-level description language. The basic geometric modelling technique is constructive solid geometry using primitive solids bounded by planes and quadrics. A variety of optical characteristics and phenomena may be specified. The scene description language allows the user to define object *classes* which may be used as if they were built-in primitives. PORTRAY uses a number of techniques, including a novel technique exploiting bounding volume coherence, to improve its ray tracing performance. PORTRAY is supported by an array of image manipulation tools which share a common image storage format.

KEYWORDS: bounding volume coherence, constructive solid geometry, illumination models, image synthesis, ray tracing.

1. Introduction

PORTRAY is an image synthesis system which generates realistic pictures of three-dimensional scenes. Scenes are described to PORTRAY using a high-level scene description language (SDL). The scene description is processed by a *scene compiler* called "PRCOMP". PRCOMP produces a file which describes the scene in a lower-level language. This intermediate file is read by the rendering program, simply called "PORTRAY", which uses ray tracing to produce an image of the scene. Figure 1 illustrates the structure of the system.

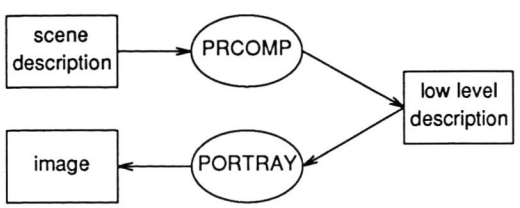

Figure 1

The PORTRAY image synthesis system is implemented in the C programming language, and runs under UNIX on VAX, Sun, and Pyramid computers.

This paper describes the geometric and optical modelling techniques used in PORTRAY, the scene description language, the processes of scene compilation and rendering, and the image format and image manipulation tools used with PORTRAY.

2. Scene Description Language

The SDL is a critical part of the PORTRAY system, since it determines the ease with which a user of the system can model the objects he wants in a given scene. With a particular SDL, some scenes may be impossible to describe, and many more scenes may be impractical to describe. The PORTRAY SDL incorporates a powerful geometric modelling technique and a variety of optical modelling techniques to give the user a large amount of descriptive power [1].

2.1 Geometric Modelling

The shapes of objects are described to PORTRAY by means of constructive solid geometry (CSG). CSG descriptions are expressions involving Boolean combinations of primitive solids. PORTRAY uses spheres, cones, cylinders, and cubes as primitive solids. These primitives have quadric and planar surfaces which make the problem of determining the intersection between a ray and the primitive quite simple. Primitives may be moved to arbitrary locations and rotated and scaled as desired. Unequal scaling may be used, for example, to turn a sphere into an ellipsoid, or a cube into an arbitrary rectangular block.

Primitive solids are combined using *regularized set operators* [2], namely union, intersection, and subtraction. The combination of two solids is guaranteed to be another well-defined solid. Any two CSG expressions can be combined using any of the three operators to obtain another CSG expression. Quite elaborate objects may easily be constructed in this way (see Figure 2).

The use of solid modelling instead of surface modelling entails some additional cost. To ray trace a solid consisting of a combination of primitive solids, we must first find the intersections between a given ray and each of the primitive solids. The resulting lists of intersections are then merged in a

way which depends on the semantics of the CSG operator. In general, we must find *all* intersections between a ray and the object, even though we only use the intersection nearest to the eye in most cases. This is because more distant intersections may affect the outcome of a merge. Ray tracing of surface models usually only requires that we find the nearest intersection between the ray and each surface. However, the additional cost of ray tracing solid models is justified by the geometric modelling power and simplicity of CSG. Solid modelling is also advantageous for simulating optical phenomena which involve light passing through the body of a solid.

Although the three CSG operators are inherently binary operators which take exactly two operand expressions, the PORTRAY SDL allows the use of "m-ary" (associative) union and intersection operations as a notational convenience. This reduces the need for the user to fully parenthesize complex descriptions to explicitly indicate the operands of every operator.

There is nothing sacred about the particular set of four primitive solids which are used by PORTRAY. In fact, the programs are designed so that a new primitive can be added simply by programming routines to find intersections, normal vectors, and texture coordinates for the new primitive, and by adding a line to a single table which links the new routines into the rest of the system. In future, primitives may be added to PORTRAY to more easily describe irregular, natural objects. CSG with simple quadric primitives is an effective means of describing most man-made objects, but the great complexity of natural objects would be modelled better by primitives such as fractal surfaces.

2.2 Optical Modelling

Shape is only one aspect of an object which must be modelled by an image synthesis system. We use the term "optical modelling" to refer to the other attributes of an object (color, texture, reflectance, transmittance, etc.) which determine how a ray of light interacts with the object. PORTRAY provides a wide variety of optical modelling techniques.

One of the most basic optical characteristics is color. The PORTRAY user may specify colors using an English-like scheme which is a simplified version of the color naming system (CNS) described in [3], or more precisely in terms of hue, saturation, and value. Both the CNS and HSV color models are generally believed to be easier for people to use than the RGB color model which PORTRAY uses for its internal computations.

The "shininess" attribute is used to specify the reflectance of a surface. The user may specify shininess in a pseudo-English fashion, using the keywords SHINIEST, SHINIER, SHINY, DULL, DULLER, and DULLEST. Alternatively, the reflectance of a surface may be specified numerically. The "smoothness" attribute controls the size of specular highlights which appear on a surface. A shiny smooth surface has smaller, sharper highlights than a shiny rough surface. The optical model simulates smoothness variations by controlling the parameter m in the Beckman function that determines the directional distribution of surface microfacets in the Cook-Torrance reflection model [4]. Extremely smooth surfaces (specified as SMOOTHEST by the user) have ray traced reflections and transmissions.

Transmission and refraction of light through objects is controlled by the "transparency" attribute. The user may specify the index of refraction and a light scattering factor for each transparent object. The light scattering factor is expressed as a distance which a light ray would have to travel through the material in order to be reduced to one-half its original brightness. The relative contribution of the reflected and transmitted rays at an interface between transparent materials is determined using the Fresnel equations of physical optics. Figure 3 is an image of a scene containing a wine glass, which illustrates the use of reflection and refraction. If the index of refraction is specified as FAKE, PORTRAY allows rays striking the surface to be transmitted straight through without refraction.

The "pure" attribute is used to distinguish between composite materials such as plastics, where the color of highlights depends only on the color of incident light, and pure materials such as metals, where the color of highlights is influenced by the body color of the material.

The "paint" attribute is used to specify a variety of texturing techniques. For each texture, the user specifies a built-in texture function and a texture color. PORTRAY interpolates between the normal object color and the paint color according to the texture function. Some of the texture functions make use of disk files of textural information. In such cases, the user specifies the texture file by name. Texture files are stored in the same format as the images produced by PORTRAY (see section 5) and PORTRAY may use several texture files during the rendering of a single scene. The scene depicted in Figure 4 makes use of nine texture functions and seven different texture files.

PORTRAY has been used to experiment with a new texturing technique called "solid texturing" [5]. Solid texture functions proved to be easy to add to the library of built-in texture functions. The left and right spindles in Figure 5 show the application of two of these solid texture functions.

2.3 SDL Statements

A scene description in PORTRAY SDL consists of a sequence of *statements*. There are several types of statements, which are described in the following paragraphs.

CAMERA, TARGET, and FOCAL LENGTH statements specify the position, orientation, and focal length of the camera which is simulated by ray tracing. Since 35 mm cameras are popular and familiar, the simulated camera is designed so that the focal lengths of the lenses used with a 35 mm camera can be used in the FOCAL LENGTH statement to obtain similar effects.

OBJECT statements specify the CSG expressions and optical attributes which describe the objects in the scene.

LIGHT, AMBIENT, and BACKGROUND statements specify the intensity, color, position, and type of light sources, and the color of the infinitely large background sphere which surrounds the scene. A direct light source may be specified as a point source, a beam of parallel rays from a given direction, or a focused spotlight with a particular concentration,

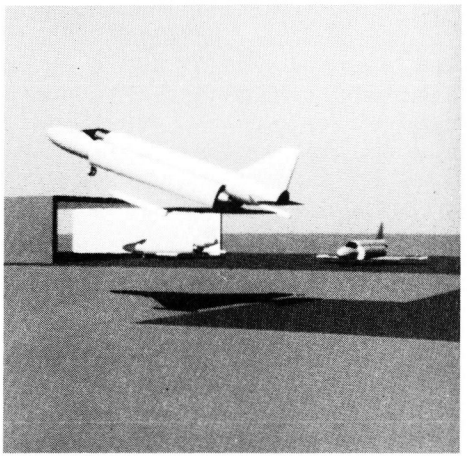

Figure 2

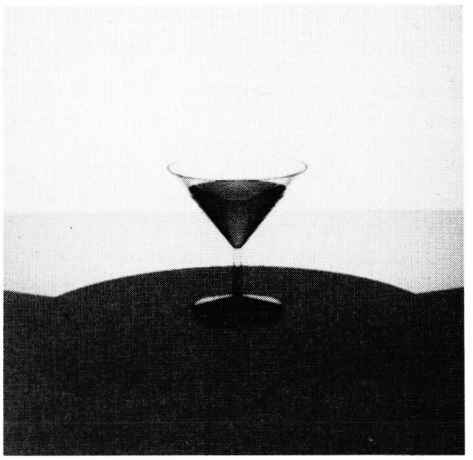

Figure 3

Figure 4

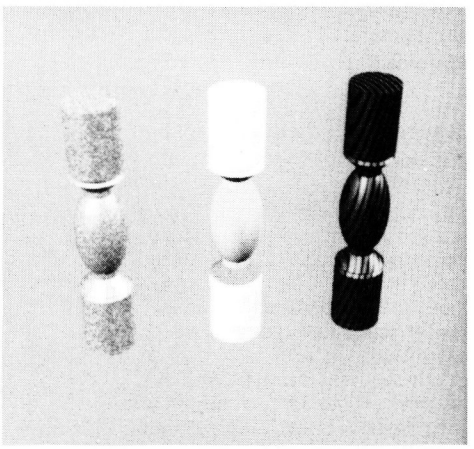

Figure 5

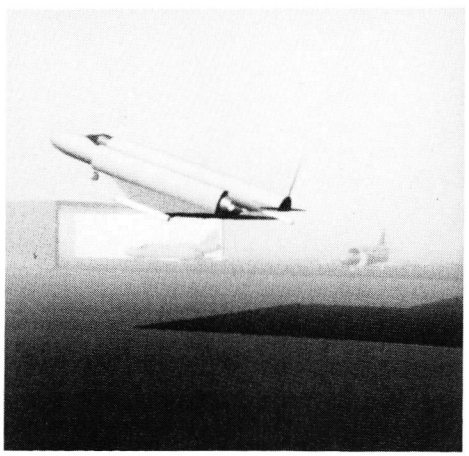

Figure 6

Note: All images were originally in color.

direction, and solid angle [6]. The user may also indicate whether or not each light source should cast shadows.

The FOG statement specifies atmospheric attenuation of light rays. Rays are faded toward the background color as an exponential function of the distance the ray travels. The FOG statement can be used to simulate day or night fog, underwater conditions, haze, or aerial perspective, depending on the background color and fog density. The image shown in Figure 6 was made using the FOG statement with a light gray background color. This image is otherwise identical to Figure 2.

The INCLUDE statement allows scene descriptions to be broken up into several files, with a main description including sub-files as necessary. This makes management of complex scenes easier, particularly in cases like animation, where object descriptions may be the same from scene to scene, with only the positions and orientations changed.

The CLASS statement allows a particular object description to be given a *class name*. Instances of the class may then be used as if they were built-in primitives. Stating it differently, the primitives are really built-in classes. A class instance may be scaled, rotated, positioned, colored, textured, etc. and may also be used as part of another class or object description.

The following is the SDL description used to produce the image in Figure 3:

```
/* Description of the wine glass ... */
Object is (smoothest shiniest trans(1000,1.65)
   white cone at (0,100,0) scale (60.5,200,200)
   rotate (0,0,-90) +
   smoothest shiniest trans(1000,1.65)
   white cylinder at (0,0,0) scale (40.5,200,200)
   rotate(0,0,90)) +
   (smoothest shiniest trans(1000,1.65)
   white cylinder at (0,350,0) scale (300,20,20)
   rotate (0,0,-90) +
   (smoothest shiniest trans(1000,1.65)
   white cone at (0,300,0)
   scale (400,300,300) rotate (0,0,90) -
   smoothest shiniest trans(1000,1.65)
   white cone at (0,320,0)
   scale (400,300,300) rotate (0,0,90)))

/* ... and of the wine itself ... */
Object is smoothest shiniest trans(1000,1.36) red cone
   at (0,320,0) scale (280,210,210) rotate (0,0,90)

/* Description of the backdrop ... */
Object is white block at (0,-20,0) scale (2000,20,2000)
Object is white block at (0,2000,-2000) scale (2000,2000,20)

/* Lighting and camera parameters ... */
Light spot (1,62) intensity 0.8 white
   at (0,1000,-100) toward (0,0,-2000)
Ambient intensity 0.30 white
Background light gray
Camera at (0, 1500, 4000) target at (0, 400, 0)
```

3. Scene Compilation

The PRCOMP program uses a LALR(1) parser, generated by the UNIX YACC utility, to parse the SDL description and build internal CSG expression trees for object and class descriptions. All instances of user-defined classes are expanded by substituting the class definition in place of the instance. The intermediate file output by PRCOMP is entirely in terms of built-in primitives. PRCOMP also expands m-ary union and intersection operators into balanced trees of binary union and intersection operators. Analysis shows that ray tracing these balanced binary trees is considerably more efficient than ray tracing the m-ary operators directly (logarithmic versus linear time complexity).

PRCOMP uses the location, rotation, and scaling information provided in the SDL description to generate a transformation matrix for each primitive (leaf node) in the output CSG description. PORTRAY uses this matrix to transform rays from the scene coordinate system to the local coordinate system of a particular primitive during intersection calculations (see section 4). PRCOMP also generates the inverse of this matrix, which is used to transform normal vectors from the local coordinate system to the scene coordinate system.

At its discretion, PRCOMP may generate a bounding volume for a given primitive instance or CSG expression subtree. PORTRAY uses such bounding volumes in order to make quick comparisons between a ray and a CSG expression so that detailed intersection calculations need not be done for rays which obviously do not pass near the object described by the expression. At present, the bounding volumes are boxes aligned with the axes of the scene coordinate system, but some experimentation with other bounding volumes [7] is planned.

4. Rendering

PORTRAY renders an image of a compiled scene description by tracing a ray from each image pixel to the scene [8]. These primary rays may generate subsidiary rays upon striking a surface which reflects and/or refracts the ray. This process may continue recursively to produce a tree of rays, whose depth is controlled by an adaptive scheme [9] and by a "hard" depth limit. Additional shadow rays are traced from each surface intersection to each shadow-casting light source, in order to determine whether or not light from the source reaches the intersection point on the surface.

Anti-aliasing is performed by adaptively supersampling when adjacent pixel values differ sharply [8]. This anti-aliasing technique may overlook very small details that "fall between the cracks", but is much less expensive than supersampling throughout the image. Recently introduced stochastic sampling techniques [10,11] are being considered as an alternative anti-aliasing scheme for PORTRAY.

Ray tracing a CSG expression [12] involves a postorder traversal of the CSG expression tree. At each internal (operator) node of the tree, lists of intersections from the left and right subtrees are merged according to the semantics of the operator (union, intersection, or subtraction). When the root of a particular expression tree (object description) is reached, the intersection nearest the ray origin is chosen as the intersection between the ray and the object. In some cases, for example, when FAKE (non-refractive) transparency is specified, PORTRAY uses subsequent intersections in the intersection list to render an object. The earth hologram display in Figure 4 is one application of fake transparency.

As mentioned earlier, intersections between a ray and a primitive are performed by transforming the ray from scene space to a local coordinate system in which the primitive has a simple, canonical form. For example, the spherical primitive is always a unit sphere centered at (0,0,0) in its local coordinate system, even though it might be positioned at (1000,2000,3000) and stretched into an ellipsoid by unequal scaling in the SDL description. It is straightforward to convert a ray into the local coordinate system using the transformation matrix supplied by PRCOMP. However, it is more difficult to convert the surface normal vector at the intersection point from the local coordinate system back to the scene coordinate system. The problem arises because angles are not preserved by unequal scaling, so that a vector perpendicular to the surface in the local coordinate system may no longer be perpendicular to the surface after the transformation. PORTRAY avoids this problem by generating three points in the plane tangent to the surface, transforming these points from the local coordinate system to the scene coordinate system, and then reconstructing the tangent plane and the normal vector in the scene coordinate system.

PORTRAY uses a number of techniques to improve the performance of the rendering process. In its simplest form, ray tracing is very much a "brute force" technique, since it exhaustively computes all intersections between every ray and every object in the scene. PRCOMP computes a bounding rectangle in image space for each object, so that PORTRAY knows which pixels may contain a direct image of a given object. PORTRAY then uses the bounding rectangles to efficiently determine which objects can be intersected by a primary ray from a given pixel. Other objects are excluded from consideration during the tracing of that primary ray.

The benefits of the bounding rectangles are limited to primary rays. PORTRAY also uses bounding boxes generated by PRCOMP to quickly exclude objects from consideration in tracing *any* given ray. If a ray does not intersect the bounding box of an object, then the ray cannot intersect the object at all. Checking a ray against a bounding box is only slightly faster than generating the intersections between a ray and a primitive. However, bounding boxes really pay off for objects with complex CSG descriptions. A single bounding box test may exclude from consideration an entire tree or subtree, thus saving dozens or hundreds of primitive intersection calculations.

Testing a ray against a bounding box is fruitful only if the test proves negative and the object within the box is excluded from further consideration. If the bounding box test is positive, the program must go on to intersect the ray with the object, so the box test is a wasted operation. (This is why it is desirable for the bounding volume to fit the object as tightly as possible [7].) PORTRAY exploits a property which we call *bounding volume coherence* to reduce the number of positive bounding box tests. Bounding volume coherence is based on the observation that rays traced from adjacent pixels follow similar paths, even down through subsidiary levels of the ray tree. Thus, there is a high probability that a bounding box test which was positive for the previous ray tree will be positive for the current ray tree. When a bounding box tests positive, PORTRAY flags it with a value indicating the ray tree position of the ray being traced. On the next ray, flagged bounding boxes are *assumed* to test positive at the same position in the ray tree, and the bounding box test is not performed. There is a performance penalty if the assumption is false, but the appearance of the image is unaffected. PORTRAY ray traces about 10% faster when bounding volume coherence is used. This is particularly interesting, since an attempt to make more general use of ray coherence, reported in [13], indicated that no performance benefit was obtained.

PORTRAY generated the image in Figure 3 at 512×512 resolution in 70 minutes on a Pyramid 90x, tracing a total of 606 thousand rays. The more complex image in Figure 4 was rendered at the same resolution in 324 minutes. Fewer rays (502 thousand) were traced in this case, because fewer pixels contained reflective or refractive objects.

5. Image Format and Tools

At a conceptual level, PORTRAY images are rectangular arrays of pixels. Each image consists of several UNIX files, including an *image description file* (IDF). The IDF is a file of ASCII text which describes the image and each of the other files which form part of the image. Table 1 lists the various files which may exist as part of an image. A particular image need not contain all of these files. The height, width, and depth of each of the image data files is described in the IDF; the data files themselves contain only the pixel intensity information. Data files may optionally be run-length encoded to reduce storage cost.

The multiple-file image representation was chosen to provide a high degree of flexibility in the manipulation of image data. For example, an RGB image with 24 bits per pixel (bpp) would be stored in three separate files, each with 8 bpp. The red, green, and blue data could be accessed separately or as a single RGB image. The format of the IDF reinforces this flexibility, since the IDF can be modified with an ordinary text editor when special handling is needed. Thus, it is not always necessary to build a new image manipulation tool, even when unforeseen needs arise.

Table 1: Image File Types

Filename Suffix	Description of Contents
idf	image description file
red	red image data
grn	green image data
blu	blue image data
cvg	pixel coverage data [15]
gry	gray-scale image data
lut	color lookup table (LUT)
enc	image encoded for LUT
log	history of image (text)

Several tools have been developed to process PORTRAY images, including the following:

- *iencode*, which generates an 8 bpp image using a given color lookup table, from a 24 bpp RGB image, using the algorithm described by Heckbert [14].

- *ilut*, which generates a color lookup table containing colors which are appropriate for displaying a given 24 bpp RGB image. The lookup table is generated from a color histogram of the RGB image, using the "median cut" color space subdivision algorithm, also described in [14]. *ilut* and *iencode* are used to prepare a 24 bpp image for display on an 8 bpp color graphics system, such as an AED terminal or a Sun workstation. PORTRAY generates all images in 24 bpp RGB form.
- *icomp*, which combines images according to a specified composition operator [15]. Figure 4 is an example of a composite image produced using *icomp*. The starfield visible through the space station windows was separately generated, and then composited with the PORTRAY image of the space station and planet.
- traditional image processing algorithms, including histogram equalization and gamma correction, are used to process texture images and to adjust contrast, brightness, and color of images prior to photographing them with a film recorder.

6. Conclusions

PORTRAY is a flexible image synthesis system which derives much of its power from a high-level scene description language. Constructive solid geometry is an excellent, easy-to-use geometric modelling technique for man-made objects, but is less appropriate for natural objects. Ray tracing is an expensive rendering technique, but is well suited to CSG models and is capable of simulating a wide variety of optical phenomena. PORTRAY incorporates various techniques for speeding up the ray tracing of CSG models, including a novel technique for exploiting bounding volume coherence. To further speed up ray tracing, we are planning to experiment with parallel ray tracing algorithms on an experimental 16-processor INMOS Transputer system, being constructed at the University of Saskatchewan.

Acknowledgements

This research could not have been performed without the support and facilities of the Department of Computational Science and the University of Saskatchewan.

References

[1] Peachey, D.R., "PORTRAY – An Image Synthesis System for Realistic Computer Graphics", Research Report 84-18, Dept. of Computational Science, University of Saskatchewan, December, 1984.

[2] Tilove, R.B., "Set Membership Classification: A Unified Approach to Geometric Intersection Problems", *IEEE Trans. Computers* C-29, 10 (Oct. 1980), 874-883.

[3] Berk, T., Brownston, L. and Kaufman, A., "A New Color-Naming System for Graphics Languages", *IEEE Computer Graphics & Applications* 2, 3 (May 1982), 37-44.

[4] Cook, R.L. and Torrance, K.E., "A Reflection Model for Computer Graphics", *ACM Trans. Graphics* 1, 1 (Jan. 1982), 7-24.

[5] Peachey, D.R., "Solid Texturing of Complex Surfaces", *Computer Graphics* 19, 3 (July 1985), Proceedings of SIGGRAPH '85, 279-286.

[6] Warn, D.R., "Lighting Controls for Synthetic Images", *Computer Graphics* 17, 3 (July 1983), Proceedings of SIGGRAPH '83, 13-21.

[7] Weghorst, H., Hooper, G., and Greenberg, D.P., "Improved Computational Methods for Ray Tracing", *ACM Trans. Graphics* 3, 1 (Jan. 1984), 52-69.

[8] Whitted, T., "An Improved Illumination Model for Shaded Display", *Comm. ACM* 23,6, 343-349.

[9] Hall, R.A. and Greenberg, D.P., "A Testbed for Realistic Image Synthesis", *IEEE Computer Graphics & Applications* 3, 8 (Nov. 1983), 10-20.

[10] Cook, R.L., Porter, T., and Carpenter, L., "Distributed Ray Tracing", *Computer Graphics* 18, 3 (July 1984), Proceedings of SIGGRAPH '84, 137-145.

[11] Dippe, M.A.Z. and Wold, E.H., "Antialiasing Through Stochastic Sampling", *Computer Graphics* 19, 3 (July 1985), Proceedings of SIGGRAPH '85, 69-78.

[12] Roth, S.D., "Ray Casting for Modeling Solids", *Computer Graphics & Image Processing* 18 (1982), 109-144.

[13] Speer, L.R., DeRose, T.D., and Barsky, B.A., "A Theoretical and Empirical Analysis of Coherent Ray-Tracing", *Proceedings of Graphics Interface '85*, 1-8.

[14] Heckbert, P.S., "Color Image Quantization for Frame Buffer Display", *Computer Graphics* 16, 3 (July 1982), Proceedings of SIGGRAPH '82, 297-307.

[15] Porter, T. and Duff, T., "Compositing Digital Images", *Computer Graphics* 18, 3 (July 1984), Proceedings of SIGGRAPH '84, 253-259.