Requirements to the Scene Data Base

Andrei B. Khodulev
Keldysh Institute of Applied Mathematics RAS

Edward A. Kopylov
Moscow State University

Dmitry D. Zdanov
Vavilov State Optical Institute, St.Petersburg, Russia

Abstract

This article accumulates requirements to Scene Data Base that can be successfully used in computer graphics for physically accurate lighting simulation. The proposed requirements to comprehensive database are a result of many years cooperation of the authors in the field of accurate lighting simulation and evolution of a number of software tools.

Keywords: *GraphiCon'98, optical simulation, 3D graphics programming, geometric database.*

1. INTRODUCTION

The standard of software interface for ray tracing is an urgent problem in computer graphics. It is very attractive to take advantage of powerful graphics hardware features for ray tracing applications with minimal programming efforts. In spite of the existence of hardware for ray tracing (for example, AR250 ray tracing graphics processors were announced by Advanced Rendering Technology Ltd.), its usage is restricted by a narrow interface. It is typical that known high-level software interfaces for ray tracing (programmable shaders) do not provide correctly the most of optical effects. First of all the problem concerns the usability of offered concept of database.

In this paper we intend to attract the attention of possible designers of the powerful interface for ray tracing and developers of the corresponding database to needs of applications different from ordinary animation software. This paper is devoted to the description of requirements to Scene Data Base (SDB) as a set of database primitives and methods used by physically based lighting simulators.

At first, let us define what we are talking about. Following Iris Inventor [1], we will consider a SDB:

The scene database is an in-memory representation of 3D objects used by program.

We would like to add here also attributes supporting, data read/save and import/export (conversion to/from other formats).

2. GENERAL CONCEPTS

The SDB keeps all info about scene. SDB should serve requests from optical simulator or other caller by returning some info about scene or making changes in scene representation.

This doc lists requirements to SDB having in mind the following operations with scenes:

- scene creation/modification;
- optical simulation in the scene;
- visualization of the scene and simulation results.

Interactive UI with SDB is not considered here, we cover only functionalities that SDB should provide.

Scene is a model of some part of real world. It consists of objects. Each object is characterized with its shape (geometry) and physical properties. SDB deals mainly with scene geometry (shapes of objects) while physical properties are important for external optical simulation program that calls SDB. SDB, however, should provide storage of physical properties as an uninterpreted by SDB data structure. This structure might reduce to a series of numbers or might be rather complex.

Objects in scene are subdivided into following classes:

(1) Real objects that are:

- ordinary objects;
- OPTOS, a special element with own ray propagation mechanism;
- light sources (LS);

(2) Imaginary objects:

- observers, a special object serves to accumulate some ray tracing information in point of ray intersection;
- OPTOS bounding volumes.

Geometry of each of these classes is described in sections below.

3. OBJECT-ORIENTED APPROACH

3.1 Open architecture

Many of those existing and new mechanisms should be designed in such way so they are easily accessible to the end user. We would like to extend functionality of our software (or even to provide such possibility for our customers). Examples of possible extensions are: new types of LS, observers of OPTOS. In particular we may want to open OPTOS interface to end user, so that user can program his own types of OPTOS. In this context, (OOO) OBJECT ORIENTED ORGANIZATION methodology to assure maximally OPEN (ACCESSIBLE) system architecture cannot be overemphasized.

3.2 Object hierarchy

A scene consists of objects. An object may, in turn, consist of other (sub-)objects, etc. We allow arbitrary level of object hierarchy. At the bottom level are elementary objects, e.g. multi-volume solids or special objects. At top level is the whole scene that may be treated as an object. Objects above the bottom level (that is not-elementary) are called compound objects.

3.3 Encapsulation

As usually with OOO, we assume that access to all objects is possible only via set of allowed operations with them. In other words, we are not interested in representation of objects in SDB, we agree to touch them only by a determined set of operations. The set of allowed operations includes, for example:

- extraction (conversion) of object geometry in(to) form of triangular mesh;
- geometric transformation (rotation, translation, etc.) of an object;
- ray tracing through the scene.

The set of operation is explained in details below.

4. RAY TRACING SERVICE

We introduce special concept of Ray Tracing Machine (RTM) to combine all kinds of ray tracing service we would like to get in SDB. RTM, for sure, will use internal data of SDB, so it will violate the encapsulation principle stated in previous section. It means that RTM should not be considered as an external program, one of SDB callers, it is actually an undetachable part of SDB.

Basically, RTM returns for a given ray the nearest point of its intersection with the scene and normal to surface at the intersection point.

However, for technical reasons we subdivide our requirements into two separate parts: requirements to SDB (that deal mainly with geometry) and requirements to RTM. Requirements to RTM are not included in this paper, RTM requirements are mentioned only when it can influence "pure SDB" topics.

5. OBJECTS GEOMETRY

This chapter describes what objects (from viewpoint of geometry) the SDB should be capable to keep. Here and below the term "object" will refer to "elementary object" (if not otherwise stated). We will consider what elementary objects can be put in SDB; compound objects can be built from them by a general mechanism (see sec. 3.2, 9.2.1).

5.1 Ordinary objects

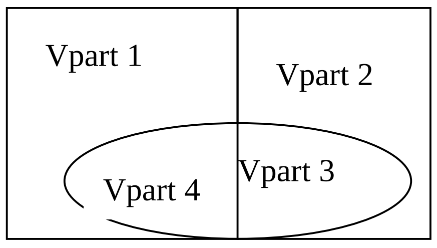
During this section we will consider "ordinary objects" that is not LS, not observer. Other, so-called "special objects", are considered in sec. 5.2.

5.1.1 Object concept

An ordinary elementary object occupies some piece of space (solid body). This body can be filled with one material (uni-material objects) or it can be subdivided into several sub-volumes with its own materials inside each one. In addition infinitely thin (one-sheet) objects are allowed.

5.1.2 Multi-material objects

The most general object (multi-material) occupies some volume subdivided into several parts (sub-volumes). We will call these sub-volumes as "volume parts" or vparts (to distinguish from simply "parts" that relate to surfaces).



Each vpart is assigned a unique vpart index that is used to extract volume attributes of the material inside. While several vparts may have the same material their vpart index should be different. An exterior of all objects is also considered as one environment vpart.

Uni-material solids consist of one vpart and are filled completely with one material (however, their surface may have different optical properties, see below). This is a particular case of general concept of object.

5.1.3 Surfaces and one-sheet objects

Objects define some surfaces in scene, namely, object and vpart boundaries. Besides that scene may contain other surfaces, those do not bound any object. We will call surfaces of last kind "one-sheet objects". When it will be necessary to emphasize the difference between one-sheet and ordinary objects (that occupy a volume) we will call the last as "solid objects" or simply "solids".

So, to summarize, both solids and one-sheet objects should be supported by SDB. Solid has some boundary (closed surface). When ray passes through boundary of solid the medium (vpart index) where it propagates changes. As contrary, intersection of one-sheet object never changes the medium. One-sheet objects, being infinitely thin surfaces are used to represent real objects thickness of which can be neglected.

One-sheet objects are restricted to lie completely in one vpart, and it should be possible to request for a one-sheet object the vpart index of vpart it lies within. (Of course, this restriction does not apply to one-sheet operands of volumetric Boolean operations to construct other objects (see sec. 5.1.6.1); only final one-sheet object should lie in one vpart.)

5.1.4 Surface part indices

All surfaces in scene are subdivided into several "parts". Each part has its own "surface part index" that is used by the caller analogously to vpart index to determine physical properties assigned to the surface (like level of surface roughness). Several surface parts can not have common part index (while physical properties of the surfaces can be the same). It is allowed that boundary between two solids is composed of several parts but not vice versa: for each surface part the solids lying at each side of the surface should not change when moving along the part.

5.1.5 Curved surfaces

Curved surfaces (solid boundaries or one-sheet objects) should be directly supported. It means that there should be some way to describe precisely curved surfaces; approximation of curved surfaces by means of polygonal (piece-wise flat) surface is not enough. The particular representation of curved surfaces used by SDB is more or less inessential: arbitrary universal (that allow to approximate any smooth surface) mechanism of smooth surface representation can be used.

For example, curved surfaces can be represented by NURBS (it provides easier conversion of geometry from external modelers that frequently use NURBS).

5.1.6 Constructuve Solid Geometry (CSG)

CSG is rather powerful tool for solid representation, in particular many type of objects used in optical design (like lenses with spherical faces) can be naturally represented by CSG.

We require that set of objects representable in SDB will be closed with respect to volumetric Boolean operations. In other words, having defined somehow two objects we should have possibility to represent their union, intersection and difference. Also some set of basic solids should be provided (like quadrics and half-space typically used in standard CSG schemes). This set should be extensible (say, we would like to include torus in future).

It should be emphasized that we do not require CSG to be used as internal representation of the object; it is not necessary. SDB can represent CSG objects in any way, but is should be capable to apply volumetric Boolean operations to objects.

5.1.6.1 Volumetric Boolean operations (VBO) for one-sheet objects

The operation of intersection of a one-sheet object and a solid has sense and it should be supported.

5.1.6.2 VBO for multi-volume solids

VBO, as they are defined, are applicable only to single-volume solids (where the function of belonging to the object can be considered as a "Boolean", that is 2-value function). For multi-volume solids the belonging functions become multi-value one. So, the concept of VBO should be appropriately extended. This extension is currently left out of this doc.

5.1.6.3 Forms of “Union”: disjoint, glue, merge

Three sorts of union operation should be supported. They are:

(1) Disjoint union (applicable to non-intersecting or touching solids). The union is formed but each of united solids retains its previous boundary. In case of touching objects this operation leads to appearance of coplanar surfaces (see sec. 5.1.9).

(2) Glue (applicable only to touching solids). In area of contact the two (coplanar) boundaries of two touching objects are replaced with a single common boundary. This boundary becomes internal boundary in new multi-volume solid.

(3) Merge (applicable to touching or intersecting objects; it is required that materials of the united solids in point of contact or intersection are the same). All common boundaries are removed; touching or intersecting vparts of different objects (having the same material) becomes a single vpart of the union.

Light interaction with material boundary is important for accurate optical simulation. Thus, we can not ignore the common boundary, as it is typically done in CSG modelers and should distinguish the three types of union with respect to their behavior at the common boundary.

5.1.7 Procedural shapes/attributes

Only procedural shapes have relation to geometry, but procedural shapes and procedural attributes are closely related (they have common description language), so they are considered together.

Procedural attributes/shapes provide a flexible way of representation of complex attribute distributions on object surfaces and complex free-form shapes. They are especially useful in design tasks where shapes/attributes should be modified by user or automatically by optimizer.

So, we need procedural shape/attributes representation. The "explicit form" requirement (described in sec. 9.1.2) applies to procedural shapes too.

5.1.8 Surface orientation

We may want to have surfaces with different properties at different sides. One example is self-emitting object (SEO) that emits light only to one halfspace. Another example: we can expect that Bi-directional Reflectance Distribution Function (BRDF) of multi-layer interference covering will be different for two sides of the surface.

So, we need to distinguish the two sides of any surface. For surfaces that are boundary of a solid there is natural meaning for the two sides: inside and outside of the solid. For one-sheet objects (and boundaries of internal vparts) such natural classification is absent. Nevertheless, we assume that all surfaces are consistently oriented, so that the two sides can be distinguished (we may assign "inner" and "outer" labels for them artificially).

5.1.9 Coplanar surfaces

It should be allowed to have coplanar surfaces in some cases. Below all such cases are presented.

It is quite common in end-user data to have two solid objects placed exactly one over the other (so that they have some common part of boundary). Such cases of coplanar (common) boundary of two solids should be treated directly.

Analogously, one-sheet object can be put exactly on surface of a solid (on its inner or outer side).

We will not, however, support two coplanar one-sheet objects belonging to the same vpart (as in such case there is no natural rule to decide in what order the ray intersects with these sheets). So, the most complex situation of coplanar boundaries is coplanarity of 5 boundaries: 2 belonging to touching solids and 3 sheet objects lying in Solid 1, Solid 2 and between them:

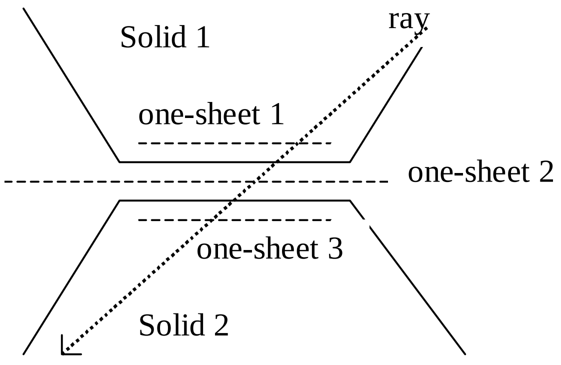


Fig. 1. Most complex allowed case of surface coplanarity.

A ray (see Fig. 1) propagating form Solid 1 to Solid 2 should intersect (in this order) one-sheet object 1, then boundary of solid 1, then one-sheet object 2, then boundary of solid 2, and finally one-sheet object 3. The distances between all intersection points are infinitely small. At second point the medium ray propagate in is changed from that of solid 1 to outer medium that surrounds both solids; at fourth point the outer medium is changed to that of solid 2; at other intersection points no medium change occur.

All other allowed cases of surface coplanarity can be obtained from the above most complex one by elimination some (any) of the five participants.

5.1.10 Light sources (LS)

Light sources are more like ordinary objects (having its own set of lighting parameters) with some peculiarity: there exist some LS types that lie in infinity. "Finite" LS like an ordinary object should be confined in one vpart.

Standard LS types are

- point;
- line (segment);
- rectangle;
- circle.

Each LS of these types should lie inside one vpart.

Infinitely distant LSes include:

- parallel
- sun (also parallel but with different set of parameters)
- sky (specified as luminance distribution over the infinitely distant sphere).

Another type of LS is important – self-emitting objects (SEO). They are some surface parts distinguished by non-zero value of the "SelfLuminance" attribute. From geometrical viewpoint they are ordinary surfaces. However, some SDB services connected with LS (say, give list of all LSes) should take SEO into account too.

As in other cases we require that SDB will be extensible to accommodate new types of LSes.

5.2 Special objects

There are some tasks where special objects are required. That is when object has no relation with media in which it lies or intersects. These objects do not influence ray traces and serve either to inform about the fact of ray intersection (clipping planes, object bounding boxes) or to accumulate some ray info in point of ray intersection (observers). Special objects may be closed or one-sheet. There are no restrictions on placement of special objects; they may intersect, include or be nested other objects. Below geometry of currently needed special objects is described (observers). However, SDB should be flexible enough to allow accommodation of new classes of special objects or more complex geometry for existing classes.

An observer has some geometry and some internal structure. SDB should support placement of observers and assignment of their parameters.

Observers can be attached to some scene surface (one or several surface parts). In such case their shape is determined by shape of the surface. Alternatively, observer can be free (not attached).

Below all current types of observers are listed. However, new types of observer will be added at user requests. So, SDB should allow extension to new kinds of observers.

Observer type	Geometry is	Comments
Camera	Viewpoint, target, screen	Ordinary camera
Section line	A segment in screen	In a sense this observer is attached to camera
Section sector	A point and 2 orthogonal vectors	
Plane	A parallelogram subdivided into grid of equal cells	
Goniometric	For free observers - sphere	
Volume	A parallelepiped subdivided into 3D grid of equal cells	Only free

6. ATTRIBUTES

Attributes mean physical properties of the medium filling a vpart, or of a surface part, or something else (say, OPTOS description can be treated as attribute). In many cases attributes are not interpreted by SDB, it is only important that user should have possibility to assign/change/request attributes for any vpart, any surface and any special object.

Below are the cases when some interpretation of attributes should be done by SDB.

6.1 Textures

Texture is a way to describe variable attributes depending on a particular point. Textures may be 2D (assigned to surfaces) or 3D (assigned to volumes).

The mechanism of texture assignment should be implemented inside of SDB.

Texture assignment should be incorporated into SDB to avoid texture distortions that would occur if we will assign textures externally with use of explicit surface representation in form of triangular mesh.

6.1.1 Orientational textures

Some tasks put into consideration anisotropic surfaces. Such surfaces have at each point some distinguished direction (tangent to the surface) that determines rotation of anisotropic BRDF. To assign anisotropic BRDF to a surface we need a tool to define this reference direction. Mathematically it is tangent vector field.

We consider this tool as a kind of mapped (2D) textures - orientational textures. The difference of all other kind of mapped textures is that orientational texture does not modify any property of the surface it is assigned to. Instead of this it introduces new property of the surface, namely, tangent vector that is used as reference direction for anisotropic BRDF assignment. SDB should support surface with orientational (and/or) ordinary textures assigned. If a surface has an orientational texture assigned, SDB should provide interrogation of the reference vector at any given point in the surface.

6.2 Procedural attributes

The mechanism of procedural attributes should allow us to specify spatial (in 2D) dependency of attributes (procedural 2D textures) as well as to include 3D spatial dependency and dependency on ray direction (procedural BRDF and luminaire distribution).

6.3 Multi-product support

Several sets of attributes may be attached to the given object.

7. OPTICAL ELEMENTS (OPTOS)

Several Plane Light Emitters and other devices (e.g. slide projector) contain special elements: plate prisms, Fresnel’s lenses, Fiberglass lens arrays (FLA). They are characterized by very complicated geometry (regular but consisted of huge number of small details) and/or non-standard ray propagation laws (in Fiberglass lenses rays propagate along curve lines). Their representation by means of usual tools is inefficient (plate prism) or almost impossible (Fresnel’s lens, FLA).

OPTOS means a special ray propagation mechanism inside some volume or in some surface. OPTOS description is not interpreted by SDB. SDB should only localize OPTOSes (that is to know the volume or surface occupied by each OPTOS) and support assignment/request of OPTOS parameters.

OPTOS can be attached to any of the following scene elements:

- vpart (it means that the OPTOS is located inside this vpart;
- bounding surface;
- one-sheet object.

So, SDB may treat OPTOS as a special kind of attributes.

Also OPTOS may contain (or may not) its own mechanism (supplied externally) of explicit form generation. SDB should support such external explicit form generators.

8. DISTRIBUTED SDB

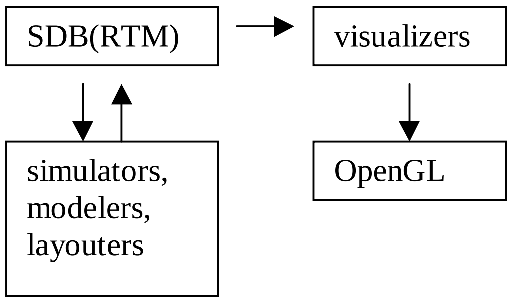
We require SDB to be DISTRIBUTED. It means that it should be possible to share the same scene between several callers located at several computers connected via network.

9. SDB SERVICES

Here we describe a set of requests SDB should serve

9.1 Visualizer services

SDB must give geometry data to visualization program in suitable form. As to this form, we choose the triangular mesh. Reasons: it is supported by OpenGL and many algorithms for triangular mesh processing are already developed.



9.1.1 Object bounding box

For each object (elementary or compound) in the scene SDB should return the object's bounding box (axes-aligned enclosing parallelepiped).

To speed-up visualization process it is helpful to know in advance what area in screen can be occupied by an object's image. Say, it allows optimizations like "Pyramid of vision".

9.1.2 Explicit form generation

It is the main visualizer service. We require for each object to obtain its "explicit form", namely, boundary representation in form of triangular mesh.

It should be emphasized that we do not request a particular form of data storage in SDB: it may be B-reps, or CSG, or something new; more probably that there should be several representations allowed. The explicit form should be generated on request.

So, the explicit form is an approximate representation of the object. We would like to control accuracy of this representation. The accuracy should be controlled by some input parameter.

Why the triangular mesh? It is supported by Open GL and many other programs.

Why accuracy control? It allows to implement standard visualization optimization technique when farther objects are visualized with use of less accurate (and supposedly simpler) explicit representation.

9.1.2.1 Projection to/from explicit form

Having internal object representation (hidden in SDB) and its explicit form, we would like to establish a point-by-point correspondence between them. Namely, for any point on object surface (found by RTM, say), we may request its counterpart in the explicit form for this object and vise versa, we would like to find the point in the object surface corresponding to a point in its explicit form. Typically this correspondence will be established along normals to object (here we mean accurate normals to real surface, not to the approximate explicit form).

9.2 Scene modification services

9.2.1 Hierarchy representation

Direct manipulations with object hierarchy should be available (combining a compound object from several other ones, destroying hierarchy, application of geometric transformations.

9.2.1.1 Possible transformations:

- translation in space;
- rotation;
- scaling factor;
- scaling rotation (a rotation to apply before scaling);
- the center point of the object for rotation and scaling;
- direct setting 4x4 transformation matrix;

Certainly the SDB should provide not only setting appropriate geometric transformation but also obtaining the transformation matrix for given object.

9.2.2 Multiple copy

One object may exist in the scene several times with different geometric transformations. That may be done as individually for each object or via group transformation - for example several rows of chairs.

9.2.3 Dynamic changes efficiency

SDB should efficiently support dynamic changes in data like adding/deleting object and changing its transformations. The last mean that if for fast RT some additional data structures as voxelization are used, then it should be effective possibility to correct them after appropriate changes.

9.2.4 Data Integrity

After data changing the SDB should detect not used data portions and delete them, correct or point as not used).

9.3 Additional SDB services

9.3.1 Correctness check

SDB should provide check of data correctness. It includes:

- collision detection (different solids should not intersect);
- closeness detection (solid boundaries should be closed);
- vpart encapsulation detection (LSes and one-sheet objects should lie inside one vpart).

9.3.1.1 Incremental correctness check

It is important to provide efficient correctness check when a local modification is done with the scene (say, one object is moved).

9.3.1.2 Optimization check

We would like to know if a special object (that can be placed arbitrary in the scene) lies completely inside one vpart. This information is helpful for optimization of optical simulation process.

9.3.1.3 Scene input/output, import/export

This requirement is not elaborated in details as it is standard one for SDB of this type. Naturally that there should be possibility to load scene from disk in memory for further operations with it and to save (modified scene) on disk.

Also we assume that SDB will support import of scenes presented in external (standard) formats, like IGES, DXF, etc., and export to these formats. Particular set of supported formats is not fixed now; we require, however, that it should be possible to extend the list of supported formats (that is to develop new converter for external formats) after completion of SDB development.

10. EFFICIENCY CONSIDERATIONS

Of course, it is necessary that SDB works efficiently, that is takes not too many memory and process requests fast. It is also important that it works efficiently for big scenes (containing thousands of objects); in particular, O(n^2) algorithms (time quadratically depends on scene size) should be avoided as much as possible.

However, overall efficiency is usually impossible and efficiency at one direction is typically achieved at the expense of inefficiency in other places. So, here we describe the most important directions of SDB use, where efficiency is especially important. They are:

- ray tracing;
- projection internal form to/from explicit form.

These are most frequently used services during optical simulation so time efficiency is especially important.

11. CONCLUSION

Authors hope this work is a contribution to creation of powerful comprehensive ray tracing interface suitable for different areas of computer graphics.

12. ACKNOWLEDGMENTS

This work is supported in part by the Russian Foundation for Basic Research under a grant entitled “Physically accurate solution of global illumination analysis” (98-01-00547) and the Integra Inc.

13. REFERENCES

[1] Josie Wernecke, Open Inventor Architecture Group. *The Inventor Mentor. Addison-Wesley, 1994.*

Authors:

Andrei B. Khodulev, a senior research scientist of M.V. Keldysh Institute of Applied Mathematics RAS.
E-mail: abkhod@gin.keldysh.ru

Edward A. Kopylov, a postgraduate student of Moscow State University.
E-mail: oek@gin.keldysh.ru

Dmitry D. Zdanov, a collaborator of Department of Optical Systems Design of Vavilov State Optical Institute RAS.
E-mail: ppodzint@admiral.ru

Аннотация

Спецификация требований к трехмерной графической базе данных.

Ходулев А.Б., Институт прикладной математики им. М.В.Келдыша Российской академии наук.

Копылов Э.А., Московский Государственный университет.

Жданов Д.Д., Государственный оптический институт им. С.И.Вавилова.

Данная статья аккумулирует требования к трехмерной графической базе данных которая успешно могла бы быть использована для физически точного моделирования в компьютерной графике. Предлогаемые требования являются результатом многолетнего сотрудничества авторов в области высокоточных методов моделирования освещенности и опыта работы с рядом программных средств.