TECHNICAL UNIVERSITY OF BRNO
FACULTY OF ELECTRICAL ENGINEERING & COMPUTER SCIENCE
DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

THESIS SUMMARY

REALISTIC RENDERING

PAVEL ZEMČÍK 1995

Author: Ing. Pavel Zemčík

Supervisor: Prof. Ing. Ivo Serba, CSc.

The thesis is based on research performed at the Department of Computer Science and Engineering, Brno, Božetěchova 2, CZ-612 66 Brno, Czech Republic in 1991-1995.

Contents:

1 Introduction

When someone says "realistic rendering of images on computer", virtually everyone who listens to him starts thinking about photographs. For this reason, realistic rendering is sometimes called "photorealistic rendering". While the idea of photographs is correct, it is necessary to define what is being rendered, how it is rendered, and not in the last place why it is being rendered. Even if it is not obvious at the first glance, rendering is not only technical task but it also involves psychology. The reason is that it is assumed that the images will be always observed by a human and the reaction of human eye and brain to the image is very important. This work is, however, oriented purely on the technical aspects of the problem.

A natural question that arises when one hears about realistic rendering on computer is: "What is this good for?". Many fields of human activities exist that can benefit from using realistic rendering. Realistic rendering may be very useful in planning of expensive objects (like, for example, buildings, big machines, etc.). In such cases we can model the object on computer, visualise it through realistic rendering, and refine it. Such sequence can prevent expensive mistakes during creation of the real object. This is an alternative to physical models in industry and in most cases the cheaper alternative. Realistic rendering can also be used in visualisation of data. It is much easier for human to analyse data that have shape and optical properties close to real objects. Let us mention for illustration few of the most frequent applications of realistic rendering:

- visualisation of data in measurement and control technology
- visualisation of physical, chemical, and other effects in nature
- rendering of parts of machines in industry
- projection of earth surface in cartography
- visualisation of human body in medicine
- presentation of data in business
- animation of 3D scenes in film industry

Only short time ago in Czech Republic computer graphics started to be freely used in all the appropriate areas of human activity. Computer graphics became widely used in various scientific, industrial, and also artistic applications even though prices of the equipment are often still high.

1.1 Current state of the art

The idea of using computer for generation images is quite natural; however, its practical realisation was reached only during sixties. As development of computer graphics is highly dependent on practical experiments, Its development also depends on development of hardware computing technology.

One of the goals of computer graphics from its beginning was producing such images of object in space that allow the user to properly recognise shape and position of the objects. The idea of creating image that would be comparable with the image of the a real scene (for example with a photograph of the scene) was till the end of the seventies considered unrealistic as it was obvious that the task is extremely demanding on both computational power and main memory of the computer. Fortunately, the development of computing technology was so fast that during the end of eighties it was possible to

experimentally visualise models of objects [Newm79] using techniques for rendering set of polygons derived from Phong's or Gouraud's methods [Gour71].

During eighties, more modern methods of realistic rendering were developed, the most successful being ray tracing [Glas89] and radiation methods [Gora84]. Such methods offer substantially better images than the older methods that approximated very roughly only the geometric optics while the more modern methods are based on approximations of laws of physics to wider extent. Results of the rendering methods at that time reached such quality that in some applications it became meaningless to improve the quality any further.

The target of current of computer graphics is not only development of methods for still higher quality of rendering with increasing precision of following the laws of physics and being still closer to simulation. Very important path of development is also development of methods improving efficiency of calculations while maintaining quality of images. The reason is that realistic rendering still is very demanding task that requires extreme computational power. If the images are to be generated in an interactive mode, very fast and powerful computers are required that are at the edge of today's technology.

Research of more efficient rendering methods helps in wider application of realistic rendering that should be possible on cheaper computers with lower computational power. The most promising methods include stochastic computations used for solution huge systems of equations in radiation methods, acceleration techniques for ray tracing, and also completely new methods like particle tracing [Patt93]. The world-wide activities in this field of research are very intensive.

Description of the current state of the art of certain details is included at the beginning of each corresponding chapters in the full thesis. Also algorithms shown in this summary are described in more detail in the full thesis.

1.2 Goal of the work

By the submitted work, I want to take part in development of the new methods for realistic rendering. To reach the goal I have defined the following subgoals:

1. Studying and analysing of the theoretical basis and methods for realistic rendering of graphical scenes.
2. Implementation of selected algorithms and checking them practically while focusing on algorithms that are independent of specialised technical equipment[1].
3. Developing original optimisation techniques and reaching improved efficiency of ray tracing.
4. Creating software system capable of editing and realistic rendering of graphical scenes and to implement such system to be able to fulfill goals 2 and 3 .
5. Publishing solutions of the above problems at the platform of reviewed conferences and journals.

[1]The goal was to develop general algorithms that are independent on type of computer and/or product of particular computer company.

2 Rendering

The way graphical scene is rendered depends on many factors, especially:

- on the type and contents of the scene
- way of representation and precision of the scene
- required quality of the image
- characteristics of the computer system
- required speed of rendering and possibility of pre-processing

Many connections exist between the listed items and by far not all the possible combinations are suitable. For example, it does not feasible to generate image with only average quality from very detailed definition of scene. Also, generating of high quality image from only roughly defined model or inadequate computer system does not have much sense.

2.1 CSG model

CSG (Constructive Solid Geometry) model of scene is based on analytic description of volume of the objects. This model is used further in the work and therefore it is described in this summary. Description of other models can be found in the full thesis. Any object that can be described as a set of points in space could be used in CSG model; however, in this thesis, only "natural objects" are considered. For example sphere (set of points belonging to a sphere) S with centre $\mathbf{c}$ and radius r could be described as:

(2.1.1) $$S = \{\mathbf{x} | (\mathbf{c} - \mathbf{x})^2 < r^2\}$$

Describing complex object only using equations (like in expression 2.1.1) could be complicated and can also lead to problems in rendering. To avoid such problems, CSG model allows building complex objects from more simple objects using set operations. The simplest objects (that are not built set operations) are called primitive objects. The examples of the most frequent primitive objects are half-space, sphere, and cylinder. The set operations that are possible to use are binary intersection ($\cap$), union ($\cup$), and difference ($\setminus$). Frequently used is also negation ($\neg$). The scene in CSG model is described as tree with set operations in its nodes and primitive objects in leaves. See figure 2.1.1 and appendix B.

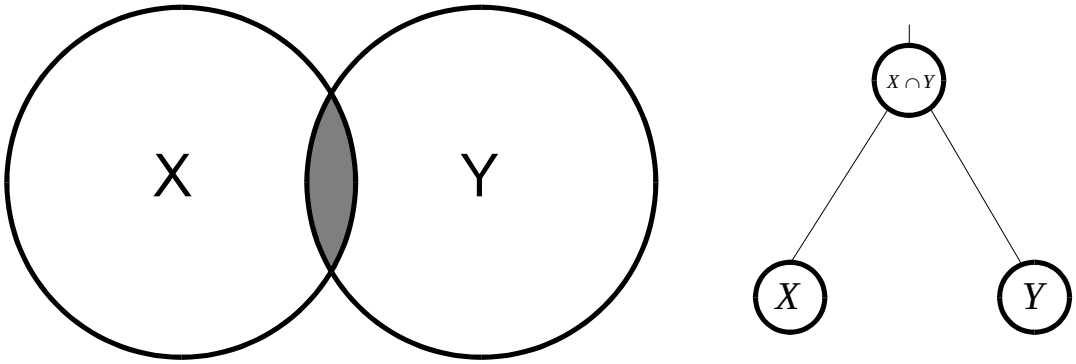

Figure 2.1.1: Object and it's CSG model (2D view)

To prevent semantic anomalies, it is necessary to precise the definition as follows:

(2.1.2) $X \cap Y = \{x | \exists \delta > 0, \forall t \in o(x,\delta), t \in X \wedge t \in Y\}$

(2.1.3) $X \cup Y = \{x | \exists \delta > 0, \forall t \in o(x,\delta), t \in X \vee t \in Y\}$

(2.1.4) $X \setminus Y = \{x | \exists \delta > 0, \forall t \in o(x,\delta), t \in X \wedge t \notin Y\}$

(2.1.5) $\neg X = \{x | \exists \delta > 0, \forall t \in o(x,\delta), t \notin X\}$

where $o(x,\delta)$ is δ-surrounding of x; therefore $o(x,\delta) = \{t | (x-t)^2 \le \delta^2\}$

The binary operations shown above can be extended to n-ary operations that allow more efficient implementation of CSG trees with many objects (real objects tend to be built from elements using repeatedly the same CSG operations):

(2.1.6) $\bigcap_{i=1}^{n} X_i = X_1 \cap X_2 \cap X_3 \cap \ldots \cap X_n$

(2.1.7) $\bigcup_{i=1}^{n} X_i = X_1 \cup X_2 \cup X_3 \cup \ldots \cup X_n$

(2.1.8) $\setminus_{i=1}^{n} X_i = ((\ldots (X_1 \setminus X_2) \setminus X_3) \setminus \ldots \setminus X_n)$

2.2 Classification of rendering algorithms

Algorithms for rendering can be classified according to different criteria [Woo90]. One of the possible criteria is how the objects are being selected for processing. Based on this criteria, three classes of algorithms can be selected.

1. Algorithms that select each object only once and process it. Typically this class of algorithms is used in graphical workstations and is supported by hardware. The advantage is very high speed if supported by hardware. The disadvantage is the limited capabilities for light sources, surface properties and shadowing.
2. Another class is formed from algorithms that determine which objects are visible in different elements of the image. After the visible object is determined, the algorithms calculate the intensity of the corresponding pixel in the image. This process repeats for every pixel of the image. Algorithms of this class can easily deal with all the geometrical aspects of the scene but they are not quite successful in soft shading. Ray tracing is the typical algorithm of this class.
3. Finally, algorithms that calculate how the energy of light is spread in the scene and only then project the scene in the image space form another class of algorithms. The energy is calculated using different techniques, for example by solving a set of equations analytically or by different stochastic techniques like particle tracing [Patt93]. This class of algorithms is the most general and is theoretically capable of rendering the finest details of the image. However, the technical limits still prevent this class of algorithms from being widely used. Also, algorithms from this class need to used some algorithm from the above two classes to project the results into the image space. All the radiation methods belong to this class.

The submitted work focuses only on the ray tracing algorithm that belongs to the second class of algorithms from the above classification.

2.3 Principle of ray tracing

The ray tracing principle has been invented by medieval painters. It comprises from creating a copy of the image that is perceived by the eye when looking in the real scene. Every element of the image corresponds to one element of a mesh placed between the eye and the real scene. In the computerised version [Gill85], [Glas89] the mesh must be dense enough to allow replacing the element by its average lightness (colour) while preserving the sufficient level of detail. See figure 2.3.1.

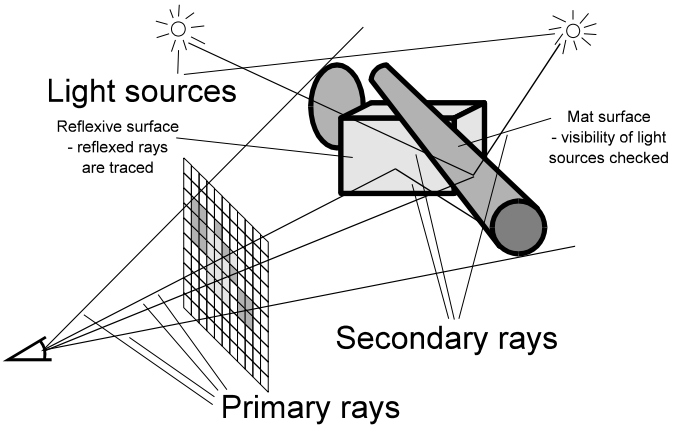

Figure 2.3.1: Principle of ray tracing

When determining the lightness of pixel, it is necessary to trace the ray starting from the eye and continuing into the scene through the pixel of interest. After it is known which object is visible through in the corresponding pixel, it is necessary to determine the pixel's lightness. This idea is not precise description of the right demand from the view point of physics but it is generally accepted as an approximation. The correct steps from the physical view point would be to consider the energy from visible range of spectra emitted in some space angle corresponding the size of pixel and only from that value to determine the lightness of pixel.

In some cases, for example if the surface of the visible object is mat, it is necessary to check visibility of light sources from the point where the ray from the eye (primary ray) intersected the visible objects. In other cases it may be necessary to send another rays from that point if the object is for example mirror. See figure 2.3.1.

Both these cases involve another rays to be traced. Such rays are called secondary rays because they do not start from the eye. The algorithm may even lead in recursive computations. If no secondary rays are shot at all, the method degrades to so called first order ray tracing sometimes also called ray casting.

After the lightness computations are finished, the last stage of the algorithm starts which is storing the image elements into computer memory and eventual displaying it on the computer screen.

3 Optimisation techniques

To be able to compare different algorithms, it is necessary to define some metrics. The suitable metrics for algorithms are time complexity and memory complexity.

Complexity is usually expressed as function of number of elements being processed by the algorithm [Broo89]. In ray tracing the number of elements may be the number of pixels being calculated or the number of graphics objects in the scene. Complexity of ray tracing algorithms is usually linearly dependent on the number of pixels; therefore, only the complexity as function of number of objects is generally considered.

Given the current state of technology, limitation for ray tracing algorithms is usually the time complexity and not the memory complexity even though the memory complexity may be also important. As the speed of CPUs is growing faster than the capacity of memory chips, it is expected that the trend will be maintained in the future. Having in mind the above facts, it is possible to consider only time complexity of the algorithms if they are suitable for "sensible" implementation.

The following symbols will be used further in the text [Broo89]:

O	class of functions with upper limit of complexity, where for $f, g \mid \mathrm{N} \times \mathrm{N}$ $\mathrm{O}(f) = \{g \mid \exists c > 0, \exists n > n_0, g(n) \le cf(n)\}$
c_m	memory complexity
c_t	time complexity
p	number of pixels in the image
n	number of objects in the scene

Only upper limit of complexity (with complexity class O) will be considered further in the text as it is the principal criterion and the complexity using real scenes is usually very close to it.

3.1 Complexity of ray tracing with no optimisation

Class of complexity of ray tracing with no optimisation can be expressed by the following equations:

(6.1.1) $c_m \in \mathrm{O}(n)$

(6.1.2) $c_t \in \mathrm{O}(n)$

Memory complexity is determined by fact that every object has to be represented in memory and no further data structures have significant impact on the memory demand. Time complexity is based on linear dependence of the maximum total time on the maximum time required for one pixel (the number of pixels in the image is constant).

The above classes of complexity look promising; however, as the number of objects in real scenes can be counted in thousands and pixels in hundreds of thousands, the total times when using ray tracing without optimisations are not satisfactory.

3.2 Optimisation of ray tracing

The ways of optimisation of ray tracing can be classified according to the part of algorithm that is affected by the optimisation (in order of estimated efficiency):

- decreasing number of the points in the image that are evaluated (undersampling)
- exploiting different types of coherence during the evaluation
- decreasing the number of ray/object intersection checks
- simplifying of the housekeeping necessary for decreasing the number of checks
- optimisation of ray/object intersection computation
- simplification of the combination of primitive objects to complex objects (in CSG)
- improving the calculations of lighting models
- optimisation of the rest of algorithm (communication, disk, etc.)

Other ways to improve the computation exist that could not be called optimisations. For example:

- parallelisation
- using the previous image to speed up the calculation of next image in a sequence

It is not possible to describe the existing optimisation techniques in this summary; therefore, only the new techniques introduced by this thesis are described.

3.3 Fast scene traversing

Increasing the speed of traversing subdivided scene is a relatively simple way how to reduce time in space subdivision techniques. Space subdivision reduces the time complexity class of ray tracing to the expressions below but adds fixed time to the algorithm that could be avoided only using very efficient traversing algorithms. The complexity of rendering regularly subdivided scene containing cube objects with the same size and orientation as the cubical grid elements is:

(6.3.2) $c_m \in \mathrm{O}(N^3 \cdot 6\sqrt[3]{n})$

(6.3.3) $c_t \in \mathrm{O}(3N \cdot 6\sqrt[3]{n})$,

where N is number of subspaces in one direction
and n in number of (cubical) elements in the scene

The above estimation is based on the fact that the number of subspace elements in the grid is N^3, that $6\sqrt[3]{n}$ cubical objects occur in each subspace element, and that general ray intersects at maximum $3N$ subspace elements.

The proposed fast traversal algorithm for uniform cubical grids [Zemc93'] supersedes other algorithms published before [Mokr92]. The algorithm generates a sequence of cubes that occur on the path of a given ray. Many efficient algorithms exist that could generate line in 3D space (for example Bresenham's algorithm or DDA algorithm [Newm79], [Soch84]). If, however, the algorithm is to be usable for subspace traversing, it must generate all the elements along the path, which is usually not the case with most general line rasterisation algorithms.

The proposed algorithm is based on the idea of choosing one "leading" axis (X, Y, or Z from the co-ordinate system of the cubical grid) for the given ray. Intersections of the ray with boundaries in the grid will occur in regular intervals if considered separately. The distances between the intersections can be measured along the leading axis and can

be marked l_x, l_y, or l_z. As rays generally do not start at the grid boundary, first distances are smaller and can be marked d_x, d_y, and d_z. See figures 3.3.1 and 3.3.2.

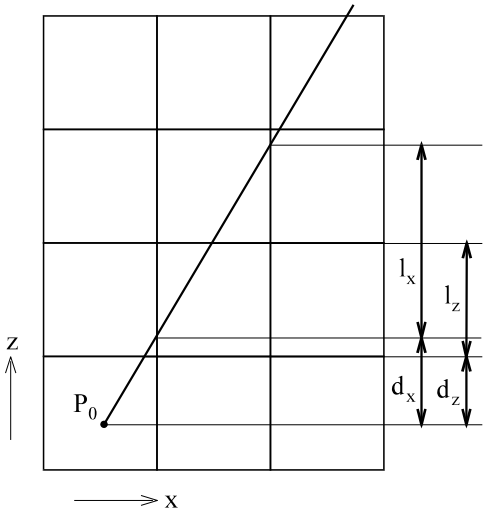

Figure 3.3.1: 2D sketch of the traversing

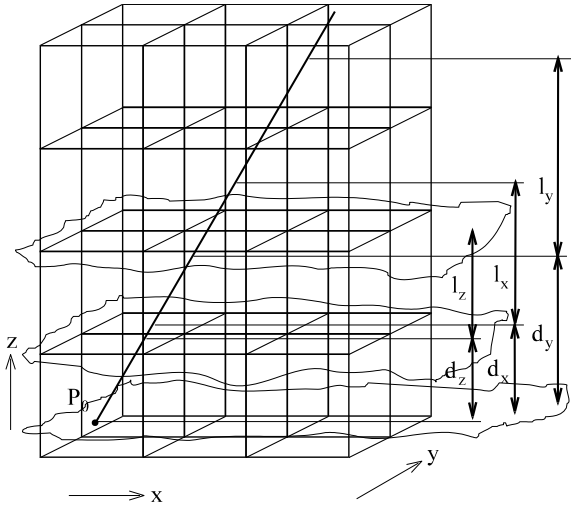

Figure 3.3.2: 3D sketch of traversing

After the distances have been calculated, it is very easy to perform traversing steps through the grid using temporary variables for the distances of the next boundaries in X, Y, and Z axes measured along the leading axis. The step is always performed in direction with the closest intersection. After the step is done, the corresponding variable is updated by adding l_i, where $\boldsymbol{i}$ is the axis in which the step was performed. This way, the step costs only 2 comparisons (to determine the closest distance) and one addition (to update the variable).

3.4 Evaluation of CSG tree in scene with space subdivision

Several CSG tree evaluation methods exist [Jans91], status tree method [Bron86], [Glas87] being considered the best. Space subdivision techniques, on the other hand, are the only techniques that really reduce the number of ray/object checks. As these two methods use different (and nearly independent) approach to optimisation, it is desirable to combine them to get advantages of both of them.

Some implementations of CSG tree evaluation in space subdivision exist that use other CSG tree evaluation methods than status tree. Good results were achieved by CSG tree pruning [Tilo84]. Pruning involves assigning each subspace CSG subtree that is derived from the original (full) tree by removing those nodes that do not change in a given subspace. The disadvantage of such approach is that in every subspace the subtree must be re-evaluated from scratch. Using status tree could besides improving efficiency of CSG tree evaluation save also these repeating re-evaluations. Unfortunately, it would be very complicated to transfer states between the subtrees. Even if some algorithms are available, they would be very time-expensive. See figure 3.4.1.

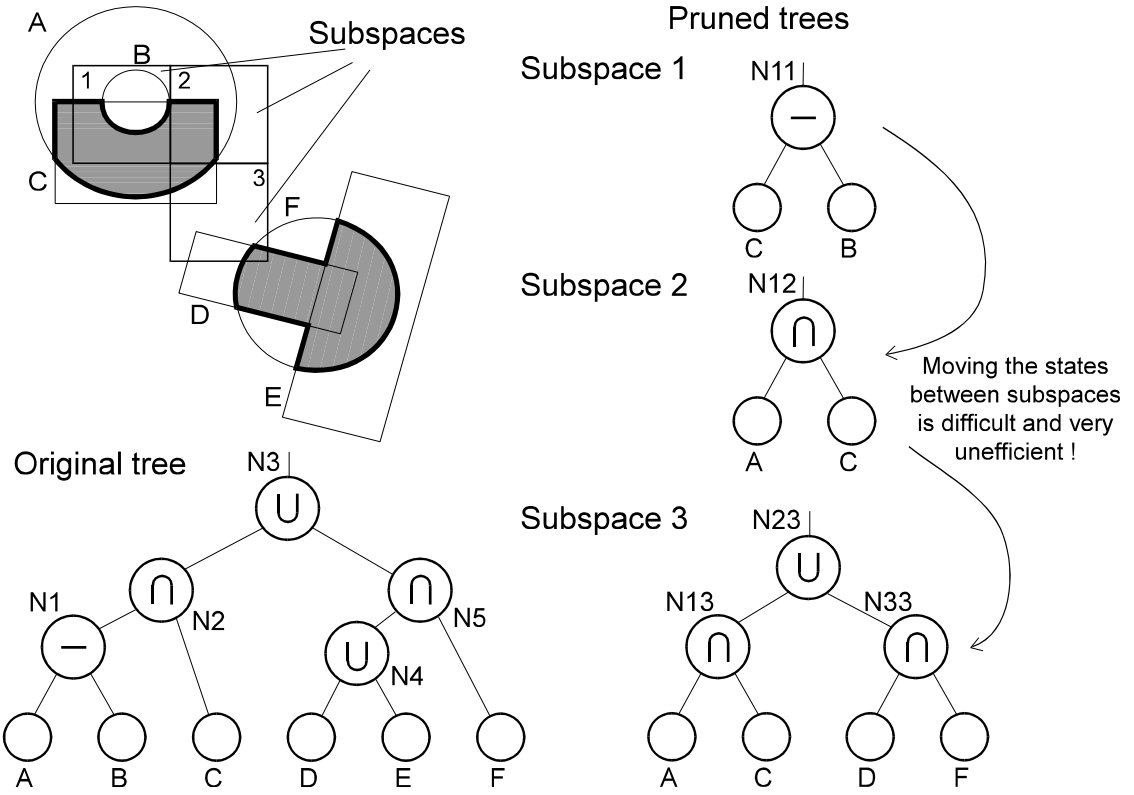

Figure 3.4.1: CSG tree pruning

The proposed new algorithm [Zemc95] is probably the first published algorithm that exploits the status tree method in the context of space subdivided scene. The algorithm is an alternative to CSG tree pruning. The CSG status tree remains as it is defined for the whole scene (unpruned). Each subspace is, however, assigned list of only those objects that are "active". In other words it means that the assigned list will not contain objects that do not occur in the subspace and those objects that contain the whole subspace.

The tree is evaluated by checking and evaluating ray/object intersections only for objects from the list assigned to corresponding subspace on the path of the ray. Only if no intersection is found, the process repeats for the next subspace along the path of the ray.

If the initial state of the state variables in the nodes of the CSG status tree is correct, the above algorithm works correctly. It is, however, the initial state of the tree that is the most expensive operation in CSG status tree from the view point of time. It would mean subspace/object evaluation, which is at least as much expensive as ray/object intersection, for each object in the tree that would render all the savings during other stages of the algorithm useless. In some publications [Glas87] it is claimed that a suitable initial state for all the objects is that the ray is outside. Such a claim is, however, not always usable.

For this reason, it is necessary not to initialise all the tree nodes but only those nodes that could affect the state of the root of the tree in a given subspace. It means that all the subnodes of all active nodes must be correctly evaluated. The subnodes are, however, not active and their state may be known in advance; therefore, it is not necessary to add

them into the subnode's list as it would mean performing unnecessary ray/object checks. It is better to create a heterogeneous lists that contain different actions on nodes.

The actions can be of three possible types:

1) Set status variable of node n to "inside" (further only set u "inside").
2) Set status variable of node n to "outside" (further only set u "outside").
3) Check (possibly evaluate) ray intersection with object o (further only check o).

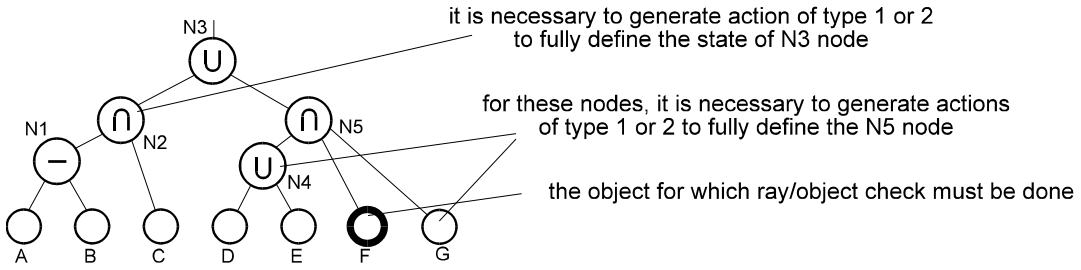

Figure 3.4.2: Adding the actions into the subspace's list

See figure 3.4.1 for the desired sequence of adding the actions into the subspace's list. The list processing for each subspace should start with actions of type 1 and 2 and only then the actions of type 3 can be performed.

The actions of type 1 and 2 are very easy to implement and they are usually in order of two or three magnitudes faster than actions of type 3. When n-ary nodes are used, the maximum number of actions of type 1 and 2 is:

(3.4.1) $n_{12} = (n_3 + d)(n-1)$,

where n_3 is number of actions of type 3 (object to check),
d is height of the tree,
and n is the maximum number of subnodes of (n-ary) nodes

Formal evaluation of speed gain of this method comparing to the traditional approach with tree pruning would be difficult although it is obvious that the gain exists (The numerical comparison of the methods depends very much on implementation.) The estimated class of complexity (6.3.3) is the same for pruning and for the new approach.

3.5 Freezing of CSG tree nodes

During the evaluation of CSG status tree using status tree method a situation may occur that state of some node is known earlier than all the subnodes are evaluated. An example is intersection node with first node evaluated to "outside" state. This fact is not used in the techniques described above. It is, however, used in boundary volume techniques and in some other methods of CSG tree evaluation.

The following algorithm [Zemc95] shows a way to improve the status tree technique by exploiting the above fact. The technique is based on the same idea as boundary volume techniques but it its implementation is different.

A situation may occur that some of the nodes of the CSG status tree received a signal (from some of its subnodes) that the subnode is always in "outside" or always in "inside" state for a given ray. In such situation the node not only passes the signal further in the

tree but also signalises to the subnodes that were not yet evaluated that they should "freeze" (that there is no reason for them to be evaluated). See figure 3.5.1.

When the frozen node in the tree receives a command to do some action (particularly to perform ray/object check), it simply does not do anything. It is clear that when next ray is ready to be evaluated, it is necessary to unfreeze all the frozen objects. As explicit unfreezing is expensive, another technique was used. Freezing of nodes is done by assigning it the number of ray for which the node is frozen. When next ray starts to be evaluated, the nodes are unfrozen automatically (the number of the ray changes).

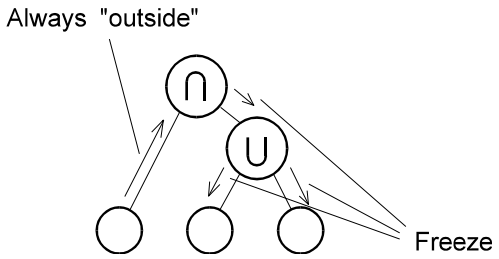

Figure 3.5.1: Freezing of CSG tree nodes

It can be seen from the description, that freezing can not only lead to improving the results but possibly even to worsening of the results as the signalisation always costs some time. Experiments have, however, shown that using this technique saves around 5% of rendering time in average.

3.6 Merging subspaces into areas

Space subdivision is an efficient optimisation technique for ray tracing. In some cases, however, finding suitable subdivision schemes may be difficult. If simple schemes were chosen, too many would have to be chosen, which means more overhead. When more complex schemes are used, also more complex methods for scene traversing that are more time consuming have to be used.

The technique proposed in this paragraph is a compromise between the two approaches. Subdivided scene is pre-processed in the same way as if uniform (cubical and dense enough) scene subdivision was used. Then all the lists in subspaces are mutually compared. If there is no difference between some of the lists, then the corresponding subspaces are merged into an area (and they are assigned the same list). This technique was probably not published but some similar techniques already exist [Devi89].

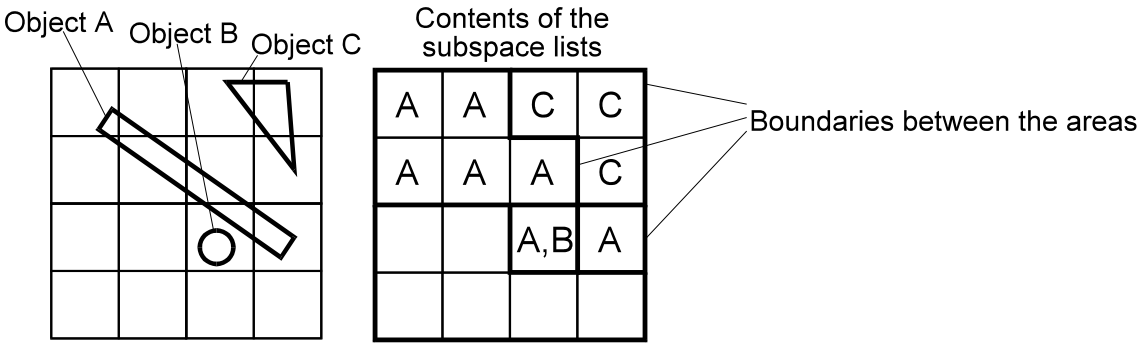

Figure 3.6.1: Merging subspaces into areas (2D sketch)

When the scene is traversed, the steps through the subspace structure are performed without any evaluation of the intersections until the area boundary is reached. Such actions can be called "moves". See figure 3.6.1.

If the above technique is used in conjunction with fast scene traversing (paragraph 3.3), the overhead necessary for subspace traversing is very little comparing to the ray/object intersection evaluation while the advantages of more complex subdivision schemes are preserved. While this technique saves only a little time and does not change the class of complexity of the algorithm, its meaning in the thesis is to show that with uniform (cubical) subdivision, which is very simple, it is possible to gain results comparable with more complex subdivision schemes.

Table 3.6.1 summarises the results from paragraphs 3.3 to 3.6 on some examples from the appendix B.

Scene	Uniform (cubical) space subdivision			Without subdivision	
	Pre-processing	With freezing	No freezing	With freezing	No freezing
Cube	8 s	144 s	150 s	130 s	>7200 s
Room	122 s	179 s	188 s	4498 s	>7200 s
Fractal	77 s	380 s	384 s	3026 s	3038 s

Table 3.6.1: Summary of results (processor i486DX2-66, resolution 256x256 pixels)

3.7 Other techniques

In some applications of computer graphics, sequences of images are being used. It is not efficient to generate each image in the sequence separately as a coherence exists between the consequent images in the sequence. The full thesis introduces a technique that significantly increases speed of generating image sequence. The technique is very easy to implement in conjunction is possible to use only if characteristics of the camera (viewer) and all the light sources do not change in the sequence.

Modern computers reach increasing of computational power by using of more CPUs in one computer. This means that if a program is to be executed faster, it should be parallelized. Generally, parallelisation may be difficult; however, most algorithms used in computer graphics can be parallelised relatively easy. Ray tracing is such a case. Calculation of value of each pixel is in ray tracing independent of calculations required for every other pixel. As images usually contain hundreds of thousands of pixels it can be expected that number of pixels will be always higher than number of CPUs and that means that parallelisation is easy.

The problematic point about parallelisation of ray tracing techniques is communication with memory. While it is possible to increase number of CPUs, it is generally not possible to have such amount of memory assigned to each of CPUs that would allow copying the whole scene to each CPU's local memory. Therefore, some caching techniques should be used to allow efficient sharing of global memory by all the CPUs while maintaining high speed of computation. The full thesis contains a study of general caching algorithms in context of ray tracing.

4 Completion of the goal

The goal that was set in paragraph 1.2 was completed. The subgoals were accomplished in the following way:

1. I have studied the theoretical background in the literature (see the list of literature). The monographs [Fole83], [Glas89] were the principal source of information. Very good sources of information were journal papers. Also consultations with Prof. Ing. Ivo Serba, CSc. (my supervisor) and others (Ing. Jiří Sochor, CSc., Doc. Ing. Jozef Honec, CSc., Dr. Alan Chalmers, Prof. Dr. Erik L. Dagless) were very important.
2. I decided to implement ray tracing algorithms that provide realistic enough image for many applications while the methods used are not dependent on any particular hardware platform. Three known methods to improve ray tracing speed were implemented: uniform space subdivision, boundary volumes, and undersampling. The description of these algorithms can be found in chapters 2 to 5 and in paragraphs 6.1 to 6.3. All the algorithms were implemented in C++.
3. The world-wide state of the art in ray tracing is very advanced. Even though, I managed to develop new original optimisation techniques. I consider the most important result of my work the technique that combines status tree approach for CSG tree and space subdivision techniques. Also the node freezing algorithm can be considered important. These techniques are described in paragraphs 3.4 and 3.5 (or 6.5 and 6.6 in the full thesis). All the techniques described in paragraphs 6.4 to 6.9 in full thesis are original.
4. The software system for scene description and realistic rendering was programmed in C++ language. I have done the implementation so that the final system can be used not only for research but also for specialised laboratory lessons at the university level. The system is currently usable on SUN, ULTRIX (DEC), MS-DOS and MS Windows platforms. For scene description I have developed specialised language PRAY (see appendix A). Development of the new language was necessary because no stabilised language for scene description suitable for CSG was available at the time the project started. The compiler of the PRAY language is, however, very simple and it can be easily changed to accept other language (it was developed using YACC - Yet Another Compiler Compiler). The PRAY grammar is built so that it would be possible to automatically create programs from graphic editor.
5. Algorithms described in paragraphs 3.3 to 3.7 were published in reviewed journals abroad and on international conferences. Precise references can be found in the list of literature ([Chal93]-50%, [Zemc93'], [Zemc94], [Zemc95]-80%[2]).

Besides working in the field of rendering algorithms I also had some activities in more general graphics and image processing algorithms, especially in scientific data visualisation ([Zemc93"]-50%), in area of basic raster algorithms for grey scale images ([Zemc93], [Zemc94]-75%), and in area in human machine interfacing and image processing ([Hole92]-33%, [Fuci94]-25%, [Zemc94"]-50%, [Zemc95]-80%).

[2]If an expression in percents follows the reference, it indicates the author's share of the publication.

5 Conclusion, further work

The theme of the submitted thesis is realistic rendering. By realistic rendering, it is possible to create an image of graphical scene. One, however, should not forget that the image must be displayed so that the user perceives with as little distortion as possible. Of great importance is also the way of using the results in application programs as I have experienced from several projects.

The images are most often displayed at computer screens. In this context, not only the maximum number of colours, geometrical resolution, and colour resolution of computer monitors are important but in some applications geometrical deformity and non-linear distortion of lightness are also important. While geometrical deformity is relatively easy to measure and compensate, the maximum number of colours and screen resolution may cause problems.

When the geometrical resolution is high enough, it is possible to compensate the low maximum number of colours and the colour resolution[3] by techniques like dithering or by filtration with error distribution [Fole83]. Such techniques are possible to use even in difficult conditions [Zemc94']. For drawing the image on the computer screen many algorithms can be used, for example filling of the closed area with pattern [Zemc93].

Along with the progress in computer technology comes the increased demand for dynamically moving image that has better potential for passing information to the user comparing to the static image. It is, however, not at all expected that static image will disappear from computer technology as only such image can be used without complex hardware and because it has well position in documentation, in distribution of the information, in scientific data visualisation [Zemc93"], and in similar applications.

Moving image can be displayed using "classical" devices like television or video devices but also directly by computer. Some application require displaying only such sequences of images that are known in advance. For those applications it is sufficient to use hardware that allows fast changing of the image on the screen. The sequence can be created in advance on some other computer than the one where the application program is to be finally used. For the target computer it means that it only has to decompress image data. Decompressing hardware is now easily available and ready for mass exploitation. Typical applications for moving image are multimedial programs.

Absolutely different situation occurs when the application program requires interaction between the user and the image. The user should be able to see more or less realistic scene and should be able to interact with it. Applications of this type are called "virtual reality". These applications have extreme demand on computer technology as the images are not only to be made on the target computer but they also should be made in real-time. Even though, virtual reality nowadays often replaces older technology wherever possible [Elli94]. A trend also exists to "merge" the expressions "virtual reality" and "multimedia" in just one term "virtual reality" as the multimedia technology is oriented towards synthetic images anyway [Rich94].

[3]On display devices with colour palette it is necessary to distinguish between the maximum number of colours and colour resolution of the palette. Typically, the maximum number of colours is given by the number of bits per pixel while the colour resolution by the width of D/A converters. If palette is not used, both terms have the same meaning.

Realistic rendering of moving images in real-time is much more demanding on the computer technology than rendering of the static images. It is also more demanding on the human visual system. It is likely that it will be necessary to search for new computational methods for efficient implementation of moving images from both the view point of technology and interaction with humans.

Suitable methods may differ from application to application because of different required "level of reality" (see paragraph 1.1). For example rendering of scientific data from the field of fluid dynamics will have very different requirements than rendering of buildings in architecture. It is, however, possible to identify the common trend. Current development suggests that ray tracing, although produces high quality image of will not be the technique for mass exploitation in virtual reality, especially for its high computational requirements. Ray tracing will probably be used as a secondary technique for Z-buffer methods with Gouraud shading (supported by hardware). Ray tracing will be used to render only those pixels that are not possible to solve with the faster Z-buffer method in good enough quality [Wojd94]. The whole images will be rendered by ray tracing only in the case when very high quality image will be needed regardless of the time. It is also expected that in the future radiation methods and particle tracing methods, which are becoming increasingly popular, will be used, most probably in combination with the above methods.

Realistic rendering in very dynamic and promising branch of computer graphics. Also for this reason I would like to keep working in this field in the future. Conditions for research in computer graphics at the Technical University of Brno (VUT) are becoming very good. One example from the last year was foundation of the university supercomputing centre in Brno (joint venture of Technical University of Brno and Masaryk Univeristy) equipped with Silicon Graphics technology. To be more specific, I would like to continue the work by the research in the field of particle tracing and by using realistic rendering in multimedia, user interfacing, and other applications.

Literature

(All the literature that was used and referenced in the full thesis is listed.)

[Hora66] Horák Z, Krupka F: Fyzika, handbook for faculties of engineering, SNTL, Prague 1966, pages 582-584 and 646

[Gour71] Gouraud H: Continouous Shading of Curved Surfaces, IEEE Transactions on Computer Graphics, no. 6, vol. C-20, June 1971, USA, pages 623-629

[Newm79] Newman W M, Sproull R F: Principles of Interactive Computer Graphics, McGraw-Hill 1979, USA, pages 20-27, 293-307

[Fole83] Foley J D, Van Dam A: Fundamentals of Interactive Computer Graphics, Addison-Wesley 1983, USA

[Drs84] Drs L: Plochy ve výpočetní technice, Matematický seminář SNTL no. 20, Prague 1984, page 52

[Gora84] Goral C M, Torrance K E, Greenberg D P, Battaile B: Modelling The Interaction of Light Between Diffuse Surfaces, proceedings of SIGGRAPH '84, ACM Computer Graphics, USA, 1984

[Tilo84] Tilove R B: A Null Object Detection Algorithm for Constructive Solid Geometry, CACM June 1984, USA, pages 684-694

[Gill85] Gilloth C, Veeder J: The Paint Problem, IEEE Computer Graphics & Applications, June 1985, USA, page 66

[Bron86] Bronsvoort W F: Techniques for Reducing Boolean Evaluation Time in CSG scan-line algorithms, Computer-aided Design, no. 18, vol. 10, 1986, Great Britain, pages 533-536

[Fuji86] Fujimoto A, Tanaka T, Iwata K: ARTS Accelerated Ray Tracing System, IEEE Computer Graphics & Applications, April 1986, USA, pages 16-26

[Glas87] Glassner A S: Efficient Boolean Evaluation of CSG Models for Ray Tracing, The Ray Tracing News, no. 1, vol. 1, September 1987, USA, pages 3-7

[Akim89] Akimoto T, Mase K, Hashimoto A, Suenaga Y: Pixel Selected Ray Tracing, proceedings of EUROGRAPHICS '89, Elsevier Science Publishers 1989, The Netherlands, pages 39-50

[Broo89] Brookshear J G: Formal Languages, Automata, and Complexity, The Benjamin Cummings Publishing Company, 1989, USA, page 254

[Devi89] Devillers O: The Macro Regions: an Efficient Space Subdivision Structure for Ray Tracing, proceedings of EUROGRAPHICS '89, Elsevier Science Publishing 1989, The Netherlands

[Glas89] Glassner A S: An Introduction To Ray Tracing, Academic Press 1989, USA

[Shin89] Shinya M, Saito T, Takahashi T: Rendering Techniques for Transparent Objects, proceedings of Graphics Interface '89, USA, Pages 173-182

[Berg90] Berger M, Levit N: Ray Tracing Mirages, IEEE Computer Graphics & Applications, May 1990, USA, pages 36-41

[Levo90] Levoy M: A Hybrid Ray Tracer for Rendering Polygon and Volume Data, IEEE Computer Graphics & Applications, March 1990, USA, pages 33-40

[MacD90] MacDonald J D, Booth K S: Heuristics for Ray Tracing Using Space Subdivision, The Visual Computer, no. 6, Springer-Verlag 1990, pages 153-166

[Stra90] Strauss P S: A Realistic Model for Computer Animators, IEEE Computer Graphics & Applications, November 1990, USA, pages 56-64

[Wolf90] Wolff L B, Kurlander D J: Ray Tracing with Polarization Parameters, IEEE Computer Graphics & Applications, November 1990, USA, pages 44-55

[Woo90] Woo A, Poulin P, Fournier A: A Survey of Shadow Algorithms, IEEE Computer Graphics & Applications, November 1990, USA, pages 13-32

[Garn91] Garnát L: Normy pro počítačovou grafiku, proceedings T2 of CAD, CAM, CAE, CIM conference, 1991 Prague

[He91] He X D, Torrance K E, Sillion F X, Greenberg D P: A Comprehensive Physical Model for Light Reflection, ACM Computer Graphics, no. 4, vol. 25, July 1991, USA, pages 175-186

[Jans91] Jansen F W: Depth-Order Point Classification Techniques for CSG Display Algorithms, ACM Transactions on Graphics, no. 1, vol. 10, 1991, USA, pages 40-70

[Serb91] Serba I: Termodynamický přístup k výpočtu osvětlení prostorové scény v počítačové grafice, proceedings of MOP '91, Československá informatická společnost 1991, pages 267-283

[Spac91] Spackman J, Willis P: The Smart Navigation of a Ray Through an Oct-Tree, Computers & Graphics, no. 2, vol. 15, 1991, Great Britain, pages 185-194

[Sung91] Sung K: A DDA Octree Traversal Algorithm for Ray Tracing, proceedings of EUROGRAPHICS '91, Elsevier Science Publishers 1991, The Netherlands, pages 73-85

[Udup91] Udupa J K, Odhner D: Fast Visualization, Manipulation, and Analysis of Binary Volumetric Objects, IEEE Computer Graphics & Applications, November 1991, USA, pages 53-62

[Herm92] Herman G T, Zheng J, Bucholtz C A: Shape-based Interpolation, IEEE Computer Graphics & Applications, May 1992, USA, pages 69-79

[Mokr92] Mokrzycki W: A New Algorithm for 3D Line Generation, Machine GRAPHICS & VISION, no. 1, vol. 1 (proceedings of GKPO '92 Naleczow), 1992, Poland, page 198

[Spac92] Spach S, Pulleyblank R W: Parallel Raytraced Image Generation, Hewlett Packard Journal, June 1992, USA, pages 76-83

[Watt92] Watt A, Watt M: Advanced Animation and Rendering Techniques, Addison-Wesley 1992, USA, pages 33-64

[Patt93] Pattanaik S N: Computational Methods for Global Illumination and Visualization of Complex 3D environments, Ph.D. thesis, National Centre for Software Technology, Gulmohar Cross Road 9, Juhu, Bombay, 1993, India

[Bado94] Badouel D, Bouatouch K, Priol T: Distributing Data and Control for Ray Tracing in Parallel, IEEE Compuetr Graphics & Applications, June 1994, USA, pages 69-77

[Elli94] Ellis S R: What Are Virtual Environments?, IEEE Computer Graphics & Applications, January 1994, USA, pages 17-22

[Gill94] Gill G W: N-Step Incremental Straight-Line Algorithms, IEEE Computer Graphics & Applications, May 1994, USA, pages 67-72

[Rich94] Rich C, Waters R C, Strohecker C, Schabes Y, Freeman T, Torrance M, Golding A R, Roth M: Demonstration of an Interactive Multimedia Environment, IEEE Computer, December 1994, USA, pages 15-21

[Soch94] Sochor J, Žára J: Algoritmy počítačové grafiky, lectures EF ČVUT, Prague 1994

[Wojd94] Wojdala A: Hardware Support for Realistic Image Synthesis and Walkthrough, proceedings of MICROCOMPUTER '94, 1994 Polsko, pages 159-170

Author's publications relevant to the thesis

[Chal93] Chalmers A, Zemčík P: Data Dependency Analysis for Efficient Ray Tracing, proceedings of SOFSEM '93, 1993, pages 33 - 36

[Hole92] Holek R, Hanáček P, Zemčík P: Real Time Transputer System for Image Processing in Airborn Recconaissance, contribution no. 41561 of the conference Modern Technologies and Industrial Reconstruction, Bulgaria, 1992

[Zemc93] Zemčík P: Scan-line Algorithm for Filling With Pattern, Machine GRAPHICS and VISION, no. 2, vol. 2, 1993, Poland, pages 143-147

[Zemc93'] Zemčík P: An Efficient Algorithm for 3D Line Generation, Machine GRAPHICS and VISION, no. 3, vol. 2, 1993, Poland, pages 231-235

[Zemc93''] Zemčík P, Serba I: Modern Methods for Scientific Data Visualisation, proceedings of SOFSEM '93, 1993, pages 93-96

[Fuci94] Fučík O, Honec J, Valenta P, Zemčík P: Cost Efficient Image Digitising and Processing Using FPGAs, proceedings of MICROCOMPUTER '94, 1994, Poland, page 225

[Zemc94] Zemčík P, Dagless E: Printing Grey Scale Images on a Fax Machine, Microprocessors and Microsystems, no. 5, vol. 18, 1994, Great Britain

[Zemc94'] Zemčík P: Differential Ray Tracing, proceedings of SOFSEM '94, 1994, pages 127-130

[Zemc94''] Zemčík P, Valenta P: Representation and Processing High Resolution Images in Computer, proceedings of Nové Smery v Spracovaní Signálov II, Slovak Republic, 1994, page 51

[Zemc94'''] Zemčík P: Image Processing Software in MS Windows, proceedings of BIOSIGNAL '94, 1994, page 66

[Zemc95] Zemčík P, Chalmers A: Optimised CSG Tree Evaluation for Space Subdivision, EUROGRAPHICS Forum, in print, accepted in December 1994, will be issued in 1995, Great Britain

Appendix A PRAY language

PRAY (Pavel's RAY tracer) language is a high level language specialised in scene description and rendering. The language is implemented in the experimental system that is one of the results of the work on this thesis.

As generating of the data structured for the scene is usually not the critical part of rendering systems from the view point of time, it was decided that programs in PRAY will be interpreted and not compiled.

PRAY supports the following data types:

- number (floating point)
- point in 3D space
- 3D vector
- intensity of light
- light source
- lighting model
- graphical object
- text string

PRAY supports the basic structured statements `if...then...else` and basic type of cycle `while`. PRAY in the current state does not support user functions and procedures.

Programs in PRAY consist of lines. The line can be or empty of it can contain statement, assignment, or language construction (`while, ...`). Programs in PRAY language can be commented. Comment begins with "//" and ends always at the end of the line. In case that one line is not long enough for assignment or statement, it can be prolonged by using "¥" in the end of the line (this is not valid for text strings and comments). In PRAY programs it is important to distinguish between small and capital letters. All the predefined structures are written in small letters.

Variables in PRAY are not necessary to declare. When a variable is needed, an assignment statement creates it with type depending on its right side. The name can contain any sequence of letters, digits and "_". If necessary, variables can be destroyed using `delete`. Example:

```
Variable  X13  _894 // Names of variables

X = 12.36       // Creates numeric variable X with value 12.36
X = 2*X+5       // Changes value of X
X = "Text"      // Redefines type and changing contents of X
delete X        // Destroys X
```

Short example of program written in PRAY. The program describes scene with only one object (red sphere):

```
RedSurface = phong({0.8,0.1,0.1},{0.3,0.3,0.3},3) // Red surface
Light = directionlight({1},norm(<1,1,1>)) // Direction light source
Centre = [0,0,0] // Centre of the sphere
Radius = 0.2 // Radius of the sphere
Sphere = sphere(RedSurface,Center,Radius) // Sphere definition
rendertiff Sphere,Light // Image rendering
```

Predefined structures in PRAY

Key words:

```
if  else  endif        // Conditional execution
while  endwhile        // Cycle
```

Predefined constants:

```
true                   // Boolean true
false                  // Boolean false
nothing          // Empty graphical object
none                   // Empty light source
```

Predefined statements (in alphabetical order):

background i,...	parallel x,y,e,p	rendertiff o,l,...
delete n	perspective x,y,e,p	quit
option o	print v	

Predefined functions (in alphabetical order):

abs(v)	planemap(l,p,v,x,y)	sin(f)
arccos(f)	dirlight(i,d)	sphere(l,p,r)
arcsin(f)	exp(f)	spheremap(l,p,r,x,y)
arctan(f)	log(f)	sqrt(f)
arctan2(f1,f2)	mirror(i)	tan(f)
color(i)	move(l,x,y)	text(l,t,p,x,y,h,f,r)
cos(f)	norm(v)	tiff(l0,l1,f)
cylinder(l,p,a,r)	one(l)	tiffbump(l,a,f)
phono(k,d,r,e)	phong(k,d,r,e)	transparent(i,f)
plane(l,p,v)	pointlight(i,p)	

Predefined operators (in order of priority):

+ -	Unary "+" and "-"
.	Geometrical transformation
^	Power
* / %	Multiplication, division, and modulo (also intersection of graphical objects and vector product)
+ -	Addition, subtraction (also union, difference of graphical objects)
= # > < >= <=	Equal, non equal, greater, less, greater of equal, less or equal
&	Intersection of graphical objects (also bitwise logical and)
| ¥	Union, difference of graphical objects (also bitwise or and xor)

Appendix B Sample scenes

CSG operations

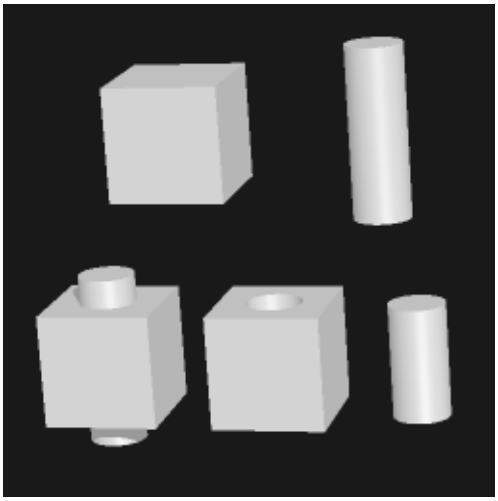

Figure B.1: CSG operations (downstairs from the left union, difference, and intersection)

Figure B.1 shows result of the following program in PRAY. The program demonstrates features of PREY language.

```
// ****************************************************************************
// *                                                                          *
// *        CSG.RAY - Demonstration of CSG operations  (c) 1995 PEZIK         *
// *                                                                          *
// ****************************************************************************

X = <1,0,0> // Calculation of directional vectors X and Y and adjusting of transformations
Eye = [2,-5,2]
Point = [0.5,0.5,0.5]
View = Point-Eye
Y = -1.1*norm(X%View)
X = 1.1*norm(Y%View)
perspective X,Y,Eye,Point

Light = dirlight({1},<-0.3,0.6,-0.4>) // Definition of light source and model
White = phong({0.6,0.6,0.6}, {0.3,0.3,0.3}, {0.1,0.1,0.1},10)

o1 = plane(White,[-0.1,0,0],<-1,0,0>) // Cube made from six half-spaces
o2 = plane(White,[0.1,0,0],<1,0,0>)
o3 = plane(White,[0,-0.1,0],<0,-1,0>)
o4 = plane(White,[0,0.1,0],<0,1,0>)
o5 = plane(White,[0,0,-0.1],<0,0,-1>)
o6 = plane(White,[0,0,0.1],<0,0,1>)
Cube = o1&o2&o3&o4&o5&o6

o7 = cylinder(White,[0,0,0],<0,0,1>,0.05) // Infinite cylinder "cut" by two half-spaces
o8 = plane(White,[0,0,0.15],<0,0,1>)
o9 = plane(White,[0,0,-0.15],<0,0,-1>)
Cylinder = o7&o8&o9

Uni = Cube|Cylinder // Three CSG operations
Int = Cube&Cylinder
Dif = Cube¥Cylinder
Ope = ((Uni.[-0.3,0,0])|(Dif)|(Int.[0.25,0,0])) // Preparation of placement
Pri = ((Cube.[-0.15,0,0.4])|(Cylinder.[0.2,0,0.4]))
```

```
rendertiff ((Ope)|(Pri)).(1.3).[0.55,0.5,0.25],Light // Rendering

// ***************************** CSG.RAY - end ********************************
```

Simple objects

Figure B.2: Rabbit

Figure B.3: Cube

Fractal

Figure B.4: Fractal (Base + 1024 distinct spheres)

Personal computer

Figure B.5: Personal computer

Glass abstraction

Figure B.6: Glass abstraction

Room

Figure B.7: Room (top view)

Figure B.8: Room (general view)

Figure B.9: Room (view 1)

Figure B.10: Room (view 2)

Figure B.11: Room (view 3)

Figure B.12: Room (view 4)