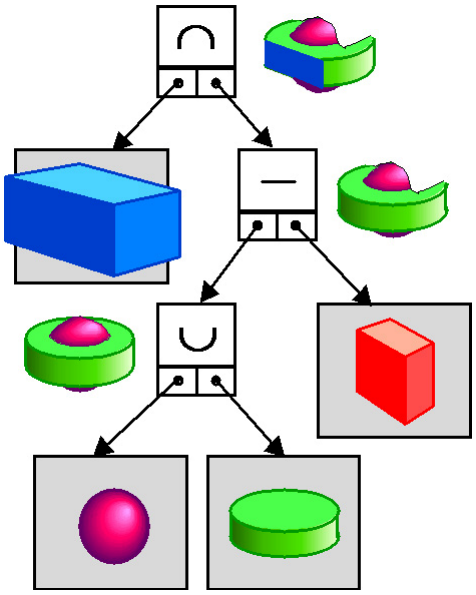
3D Object Representations

This section discusses concepts and data structures for representing three-dimensional objects in the computer. In the section "Graphics Primitives" we have discussed polygonal representations of objects (boundary representation = BRep) as a concept and the polygon-lists as the proper data structure. In the following, other types of representation are described.

Constructive Solid Geometry (CSG)

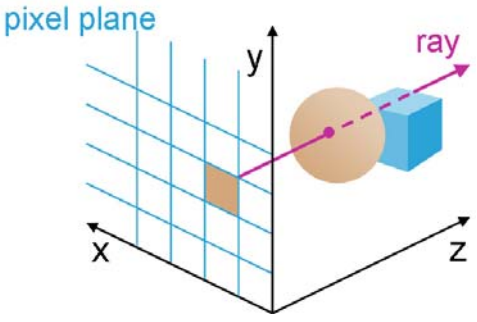


Using CSG, objects are constructed by applying set-operations to three-dimensional primitives. The operations are arranged in a hierarchical data structure, usually called a CSG-tree, which is a graph without cycles. The primitives are geometric shapes like a sphere, cube, cylinder or tetrahedron. These primitives are combined with the operators union, intersection and difference. Since all primitives are consistent, which means that they have no holes in their surface and a well-defined interior, and the operators can create only consistent objects from these consistent parts, all CSG-objects are always consistent.

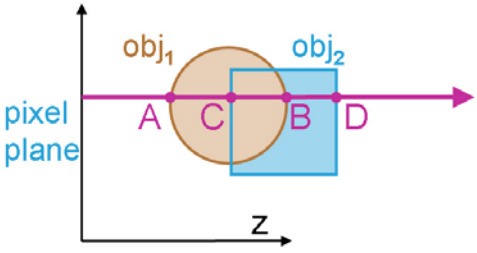
Furthermore, in a CSG-tree every node contains also transformations, in the form of matrices, which describe which transformations have to be applied on the attached sub-tree. This allows primitives to get new shapes and new positions (e.g. an axis-parallel unit-cube can become an arbitrarily positioned cuboid), and that also each complex object can be translated, scaled, rotated, etc.

Advantages of CSG-objects are the exact representation (a sphere is really a sphere!), the low memory-consumption and the simplicity of transformations. On the other hand, the main disadvantage is the considerably more costly computation of images, meaning that the rendering is more complex. For this, either the data structure is converted into a BRep and rendered in a conventional way, or ray-casting or ray-tracing is used for direct image-creation.

Ray-Casting of CSG objects

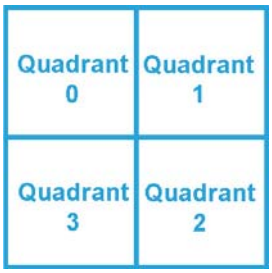


The most common method to render CSG objects is ray-casting, which calculates the image pixel by pixel. For each pixel a ray is cast in the view direction, which is then intersected with all objects in the scene. The foremost intersection point is then the object point which is visible at that pixel and therefore the pixel gets the color of that object. For CSG-Trees, this calculation is performed recursively:

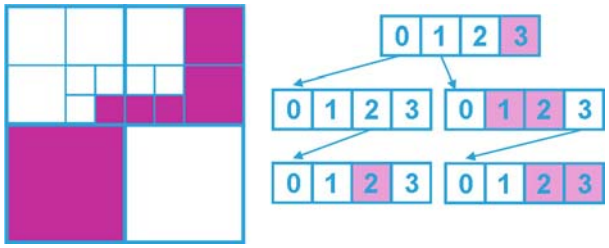


- at **leaf nodes** the calculation of all intersection points is simple,
- at **intermediate nodes** the intersection point lists of the two child nodes are combined by an operator accordingly:
from the lists (A,B) and (C,D) in the example to the right we get
with union the list (A,D)
with intersection the list (C,B)
with difference the list (A,C)
- at the **root node** the first point of the combined intersection point list is chosen.

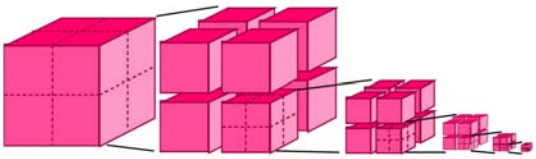
Quadtrees and Octrees



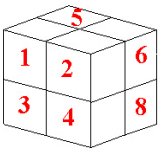
A **quadtree** is a data structure designed for the representation of arbitrary two-dimensional structures. Wherever the information is still too complex to be simply stored, the relevant part is divided in four quarters, otherwise the simple information part is stored in a leaf node of the quadtree. Each part of the image corresponds to a node of the tree, where each node has (at most) four children ("quadtree").

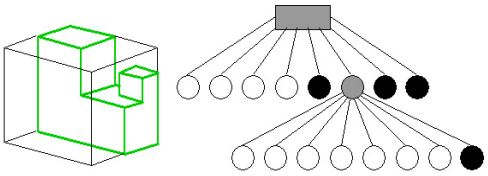


The image to the right shows a simple quadtree representing a simple two-color image. The root node corresponds to the whole image, the nodes in the second level correspond to the two upper quarters of the image and the last two nodes correspond to the two regions with the highest resolution.



An ***octree*** is the extension of the quadtree concept to three dimensions. An arbitrarily shaped object (or even a whole scene) within a cube is represented in a way that "simple" sub-cubes (empty or completely inside an object) are represented by leaf-nodes; whereas complex sub-cubes (all others) are subdivided into eight smaller sub-cubes (octants) to which the same rules are applied (recursively). Thus we end up with a tree in which each node has 8 children ("octree"). We also stop subdividing further when a sub-cube is smaller than some threshold (e.g. a thousandth part of the root node's cube-size); in that case the tree's node gets the best available simple information. That happens at sloped surfaces sooner or later, and these border-cubes must then be declared as either inside or outside. The example to the right, with only two levels, also illustrates the problem of limited spatial resolution.





The advantages of octrees are that they can represent arbitrary shapes and that we can quickly analyze what is present at a specific position in space. The disadvantages are: imprecise representation, high storage demands and complex transformation operations. Octrees, like quadtrees, are handled recursively. Thus, set operations are simple, however geometric transformations (apart from exceptions) are burdensome because the octree must be regenerated completely. On the other side, using re-writeable memory, the rendering of octrees is simple:

```
if node is simple
then draw node (i.e. do nothing if node is empty)
else recursively call the 8 octants from back to front
```

Other Object Representations

There are many other object representations and corresponding data structures which have been developed for very specific objects and applications. Some of them are BSP-trees, fractals, shape grammars along with procedural models, particle systems, physically based models, three-dimensional volume data, and several others. In the next section we will take a closer look at curved surfaces and freeform surfaces. The data structures will be treated in advanced courses.