PARALLEL RENDERING

Paul Coddington, Alvin Leung and Pat Angeles

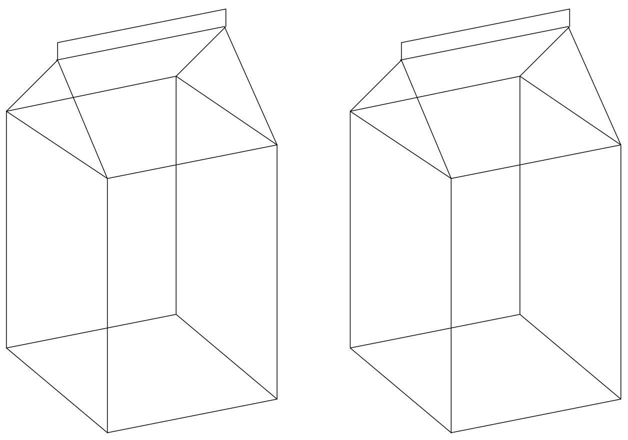
Northeast Parallel Architectures Center

October 1994

OVERVIEW OF COMPUTER GRAPHICS

Computer Graphics

The goal of computer graphics is to show a virtual world. This virtual world is composed of many different objects. A rendering program takes the computer models of the objects and converts them into an array of pixels that are rendered on the computer screen to give an image of the virtual world. The virtual world in this talk consists of two milk cartons.



Objects

Object Representation

How the objects are represented in the object database has a major impact on the rendering performance, the quality of the generated image and the time needed to construct the virtual world database.

The milk cartons can be represented in two ways. In the first method, the milk cartons are represented as a milk carton object. The scene designer need only specify the dimension, the texture map and the color. Each object in turn is composed of number of rectangular and triangular 3D sub-objects. The sub-object is positioned relative to an anchor point of the object.

These objects are analytically defined, so there is no limit on resolving the details of the object. The resolution of the object is bounded instead by the surface texture map.

Polygon Representation

The second method of modeling an object is by polygons. Generally, 3 or 4 vertex polygons are used. Triangles are more often used because all vertices of a triangle are coplanar, so the polygon will have only one normal, which greatly simplifies some rendering computations such as shading. The scene designer will construct an object one polygon at a time, and then define the surface texture and the color for each of the polygons.

The motivation for defining an object by polygons is flexibility of the object representation. Each polygon is represented by a fixed set of points and edges joining the points. These points can be arbitrary positions in 3D space, so deforming an object is a relative simple task (e.g. a filled milk carton's side can be slightly warped).

The Rendering Process

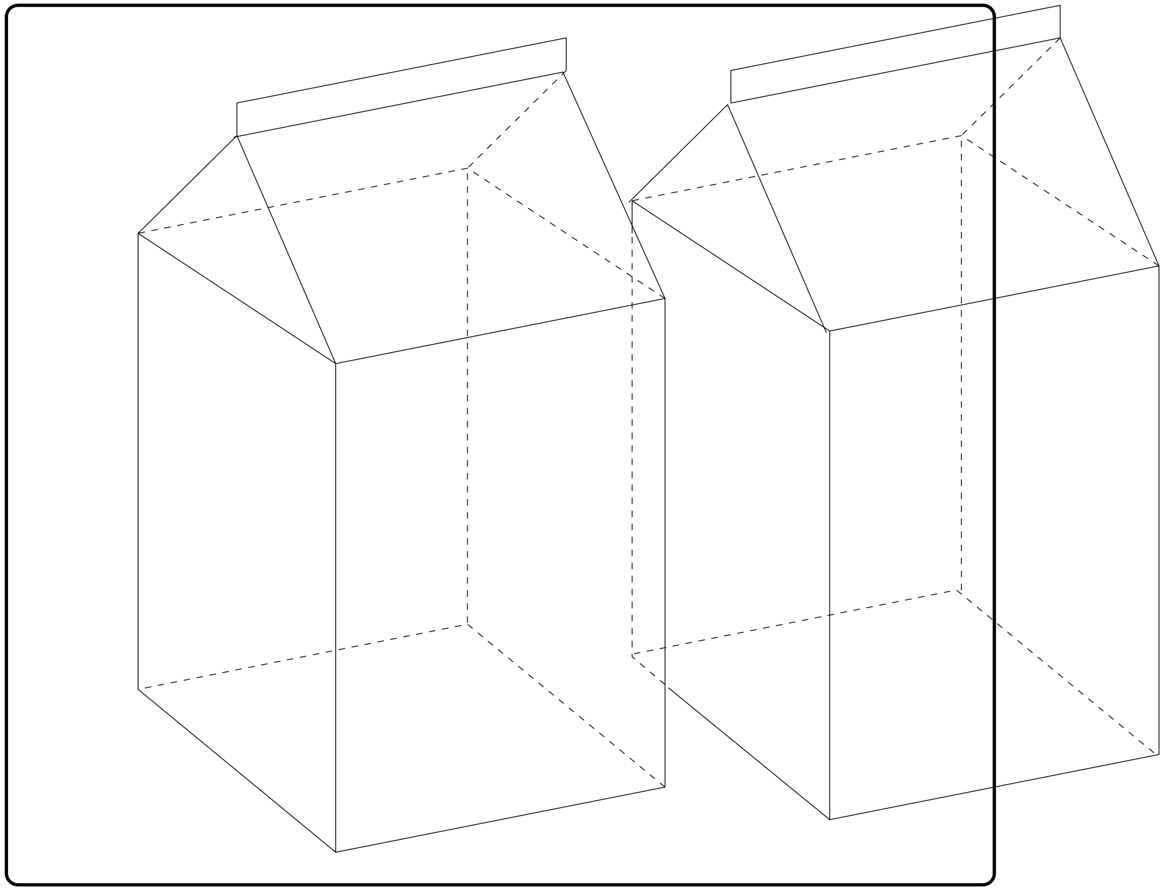
Rendering a 3D image requires the following computational procedures:

1. Transformation
2. Hidden surface removal
3. Clipping
4. Rasterization

Transformation

The positional placement of the objects relative to the viewer is done by transformations. The transformation process includes translation, rotation and scaling. Each individual transformation can be represented as a 4x4 matrix (one extra dimension for non-linear perspective scaling). These matrices can be multiplied together to form one final transformation matrix, which is used to transform the objects. Transformation is used to place the virtual world in the viewer's perspective.

An analytically defined object is moved by transforming the anchor point of the object. The transformation can therefore be done to the object as a whole, and thus takes time O(#objects). Transformation of a polygon-based object requires computation at each vertex, and thus takes time O(#vertices).



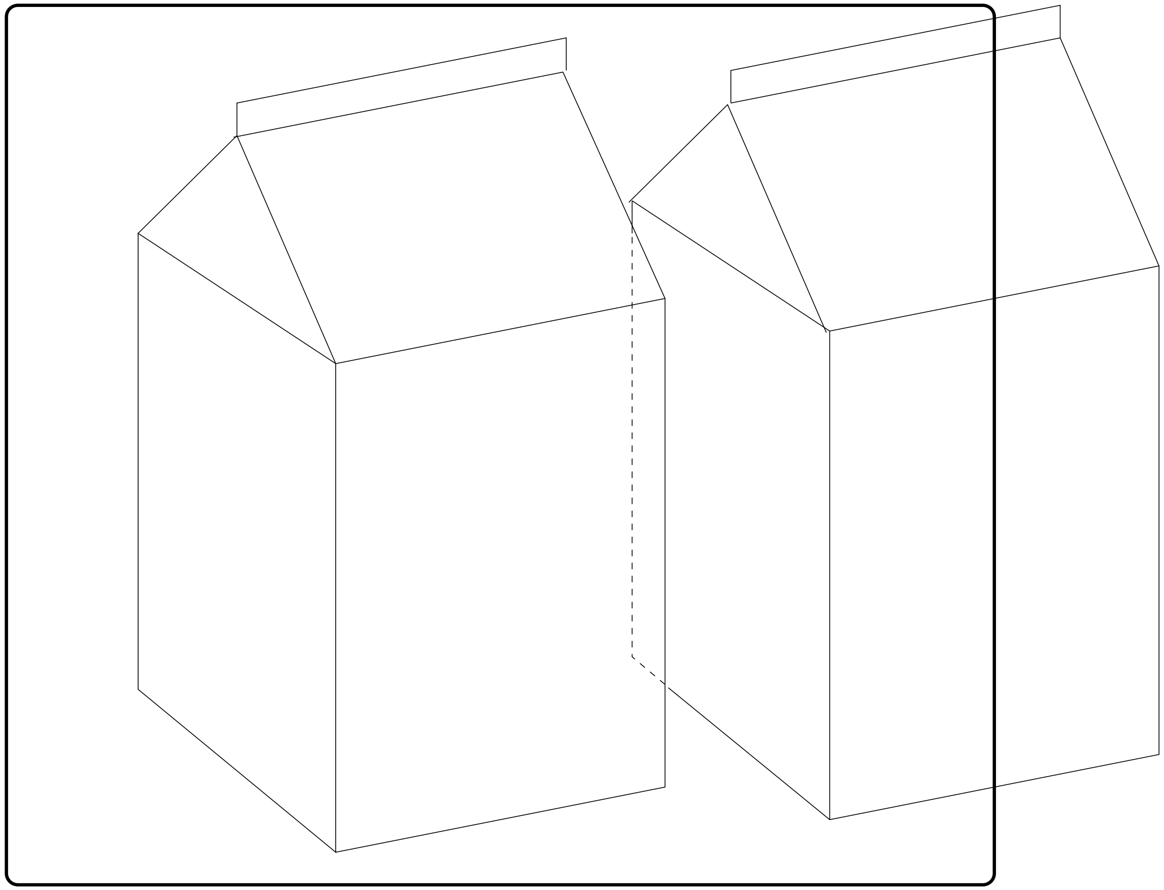
After transformation

Hidden Surface Removal

Even when an object is within the field of view, it may still not be visible to the viewer – for example, an object located behind a closed door. These have to be eliminated from the object rendering list.

Hidden surface removal could also be done later in the rendering process (during rasterization), by just having objects closer to the viewer overwrite objects further away. However, this wastes considerable computation in unnecessary rendering of these hidden objects. It is cheaper to identify and remove them earlier in the rendering procedure, when the work is proportional to the number of objects rather than the number of pixels.

For polygon-based rendering, hidden surface removal can be done very fast by just culling the back-facing polygons.

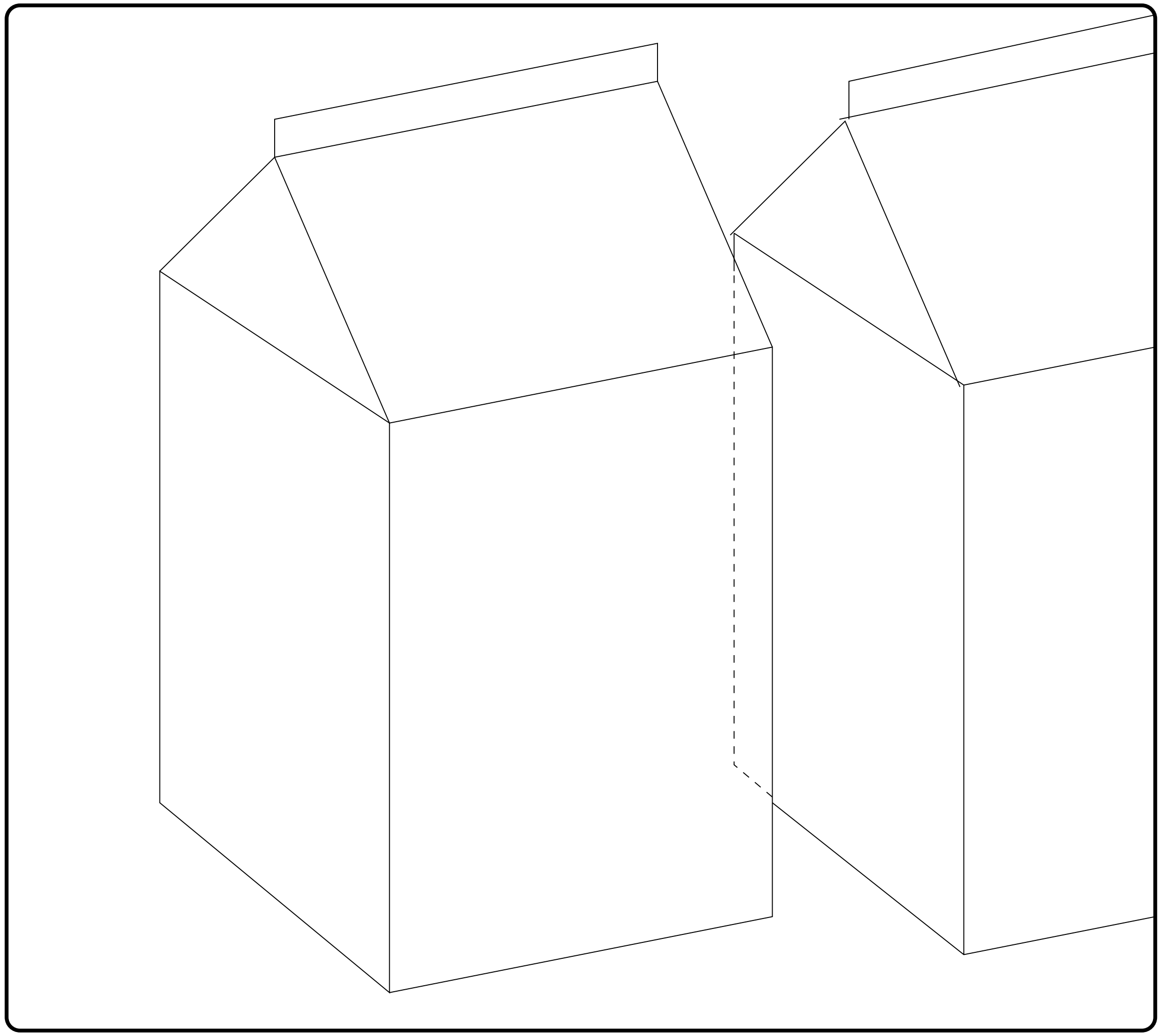


After hidden surface elimination

Clipping

Clipping is done to eliminate objects (or parts of objects) that are outside the field of view, for example, objects behind the viewer. Clipping also speeds up the rendering process since the eliminated objects do not need any further processing.

Clipping can also be used to eliminate unwanted objects, for example those that are too far away to have enough effect on the scene to warrant the amount of computation required to include them.



After clipping.

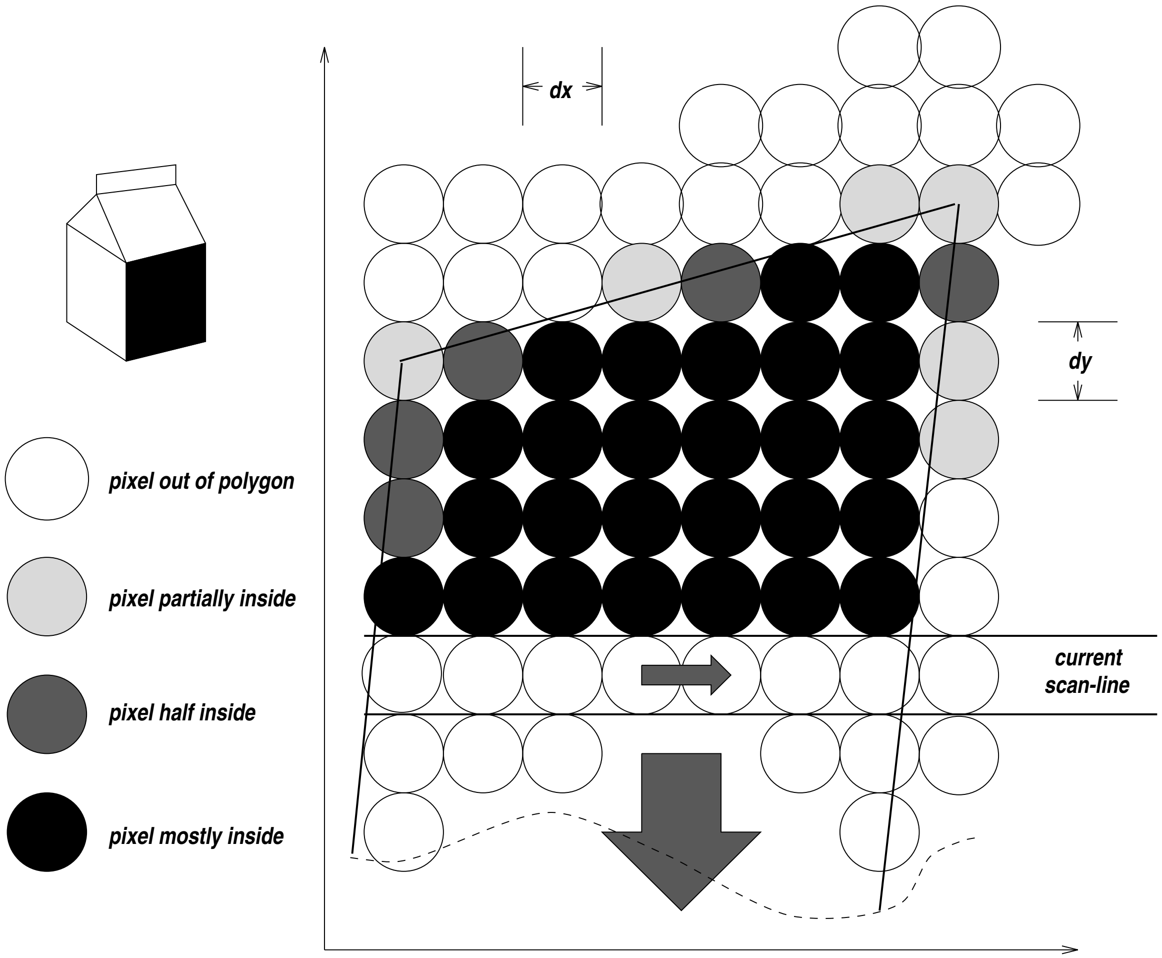
Rasterization

Rasterization is done to transform the data represented by the object database into a pixel-based image. Texture mapping, color computation, shading and anti-aliasing are done at this stage.

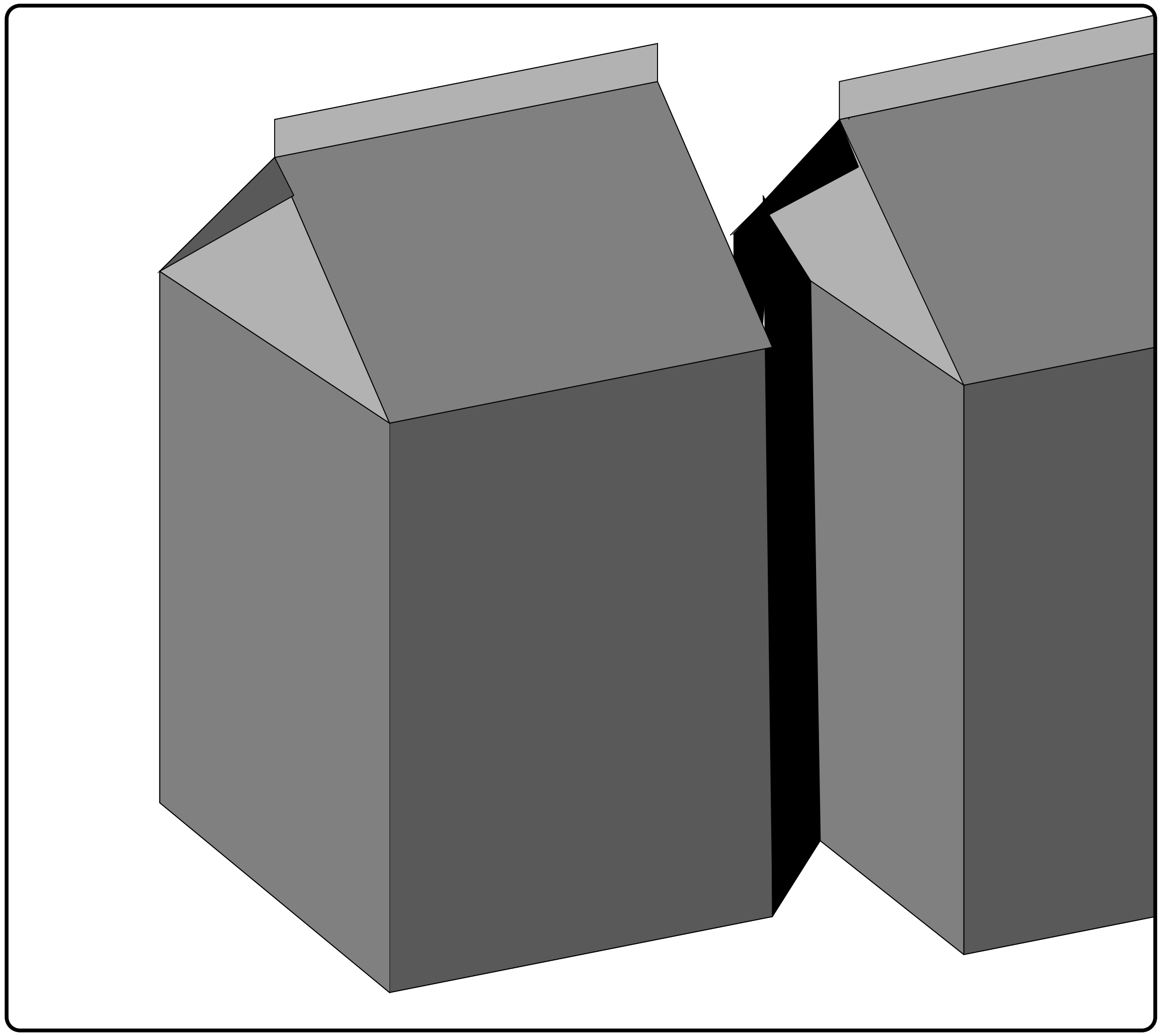
For a polygon based object, a *scanline* algorithm is commonly used to render the final image. The pixels are scanned a line at a time and their color is determined from the polygons that contribute to that part of the screen.

The colors of the polygons are specified either at the vertices, in which case interpolation is used to find the pixel color, or by a texture map, which can be transformed into the appropriate pixels.

A *Z-buffering* algorithm is used to ensure that objects closer to the image (along the z direction) are rendered last, so that partially obscured polygons are overwritten (as they should be).



Rasterization of a polygon.



After rasterization

PARALLEL RENDERING

Parallel Rendering Algorithms

The rendering pipeline remains the same as the sequential algorithm, with polygon calculations (transformation, clipping, hidden surface removal) and pixel calculations (rasterization).

Both types of calculation are completely independent, so can trivially parallelize over polygons or pixels. However there are the usual problems of load balance and communication overhead that need to be overcome to get good performance, which is particularly crucial for real-time animation.

One advantage in doing animations is that successive frames are very similar. This allows dynamic load balancing, since the computational load for current frame can be used to load balance the next frame.

Problems with Parallel Rendering

Load balance is needed for the polygon calculation. A scattered or cyclic distribution may work well, or else do dynamic load balancing by using data on work distribution for previous frame to load balance the next frame.

Parallel overhead is incurred due to the two different data decompositions used (polygon domain and pixel domain). Since polygons are irregular they will overlap pixel regions on different processors. Polygons can be cut at processor boundaries.

Communications overhead must be minimized since a frame has to be rendered and displayed every 40 msec for real-time animation. Cannot afford too much communication if latencies and communications times are in the msec range. Overlap calculation and communication?

Ray-Based Algorithms

There are also parallel rendering algorithms based on raytracing and raycasting, which follow individual light rays to or from the viewer. These are inherently parallel (each ray is independent), however they require a copy of the entire dataset on each node, or else lot of costly data movement.

NASA JPL has developed a parallel "ray-identification" program using a method of this kind. This independently transforms pixels into the view plane, and uses Z-buffering for hidden surface removal. Scattered data decomposition is used for load balancing.

These methods produce high-quality results but are very computationally intensive. The JPL program has been used to make movies of simulated flights over Venus, Mars, and the Earth. It can be run interactively (1.5 fps) on a 256-node Intel Paragon.