TERRAIN RENDERING FOR A GEOGRAPHICAL INFORMATION SYSTEM

Paul Coddington

Northeast Parallel Architectures Center

November 1994

THE INTERACTIVE JOURNEY

The Living Textbook

The Living Textbook Project is a joint project between the Northeast Parallel Architectures Center (NPAC), the School of Education at Syracuse University, NYNEX, and Columbia University, on the application of high performance computing and communications (HPCC) to K-12 education.

Several pilot projects involve designing multimedia educational applications, including interactive simulations, which will be hosted on computing facilities at NPAC and delivered to schools in Syracuse and other parts of New York state via the NYNet fast ATM network. This is a prototype for the use of the National Information Infrastructure (better known as the Information Superhighway) for education.

The 3D Interactive Journey

As a part of the Living Textbook project, we are developing a flight simulator interface to geographical data of New York state. Students will be able to "fly" around a rendered 3D view of the state that is interfaced to a Geographical Information System, so that they can select landmarks and cities and obtain multimedia information about these places.

Currently high-quality real-time 3D terrain rendering is not feasible on single processor workstations, and certainly not on machines such as Macs that are likely to be used in schools. So we are aiming to interface existing 3D GIS software running on a Mac with a parallel terrain renderer running on a high-performance parallel computer connected via a fast ATM network.

Geographical Information Systems

The United States Geological Survey (USGS) defines a Geographical Information System (GIS) as

> "a computer system capable of assembling, storing, manipulating, and displaying geographically referenced information, i.e. data identified according to their locations."

Many types of geographical data can be used:

- Elevation
- Satellite image
- Digital line graph
- Land use / Land cover
- Geographic place names

Integrating these different data sets is a non-trivial problem.

DIGITAL TERRAIN DATA

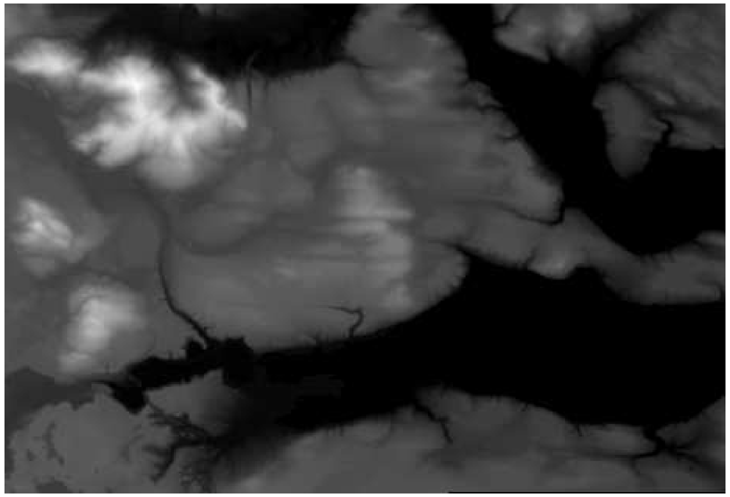
Elevation data

Elevation data is available from USGS as a Digital Elevation Model (DEM). This is obtained by digitizing contour lines on USGS topographical 1:250,000 scale maps. The data is set up as points on a grid covering a 1 degree quadrant with 90m resolution. 30m resolution data is being developed, but currently is only available for about half the US.

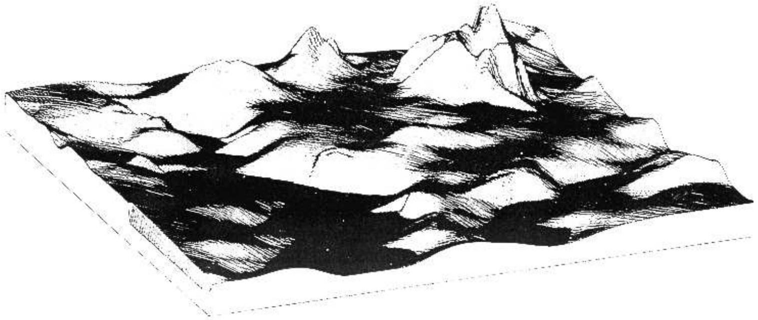
Data for New York state requires about 500 Mbytes, for the entire US it is about 10 Gbytes. This will increase 9-fold for 30m resolution data.

The US Defense Mapping Agency has 90m resolution data for the entire world, but it is classified. The only available world-wide data has 1km resolution, which is the same as is available for Mars!

Example DEM data



2D image, with brightness increasing with height.



3D image with shadowing.

Digital Elevation Model (DEM) data for part of the Cascade mountains in Washington state.

Satellite image data

Landsat satellite data of will provide realistic colors and texture for the polygons, with a resolution of 30m.

Data first has to be preprocessed to correct for distortion caused by for varying elevations, using the DEM data, so that it becomes like a 2D map. The satellite image data can then be "draped" over the DEM data to make a 3D image.

Satellite image data at better than 1km resolution is available for Mars and Venus from JPL.

Example satellite image data

Landsat image of Cape Cod

Digital Line Graphs

Digital Line Graphs (DLG) are made up of vector data marking objects such as roads, waterways, city, county, state and country borders.

Digital line graph of roadways

Other data

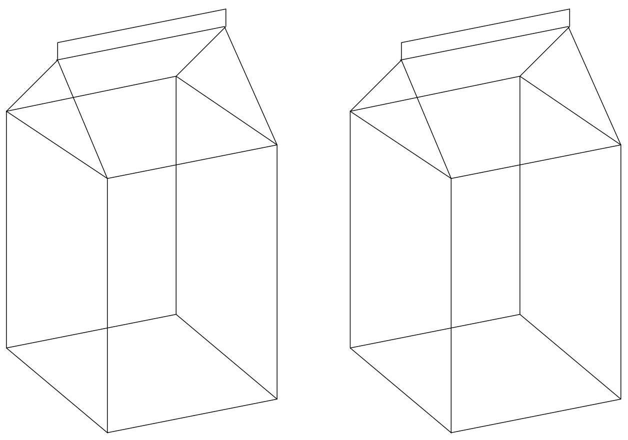
Land use / Land cover data is available from USGS. Areas are classified into 92 categories of land use (human activity such as residential, agricultural, etc) and land cover (vegetation, water, etc).

Geographic place name data is available from USGS, listing all known places, features and areas in the US that are identified by proper names. The database contains over 2 million place names referenced by geographic position.

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

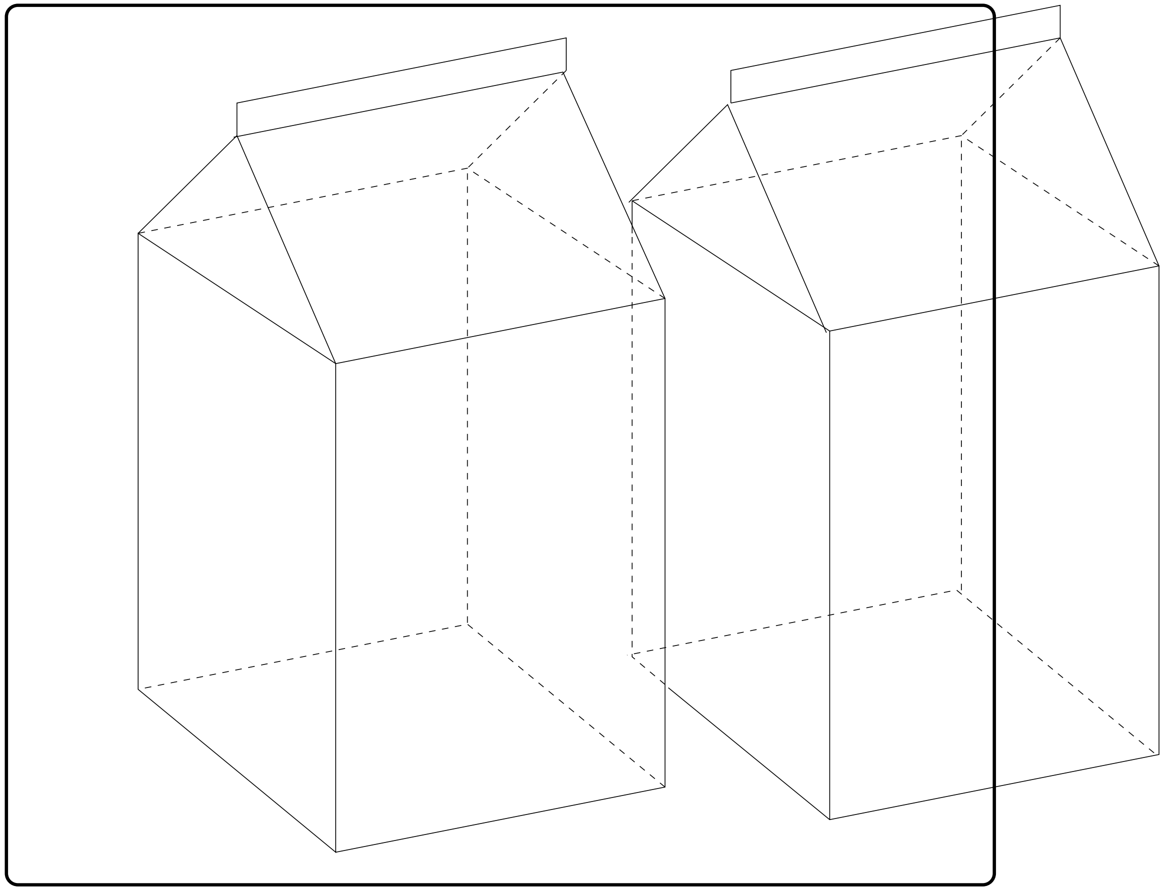
Rendering a 3D image requires the following computational procedures:

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).



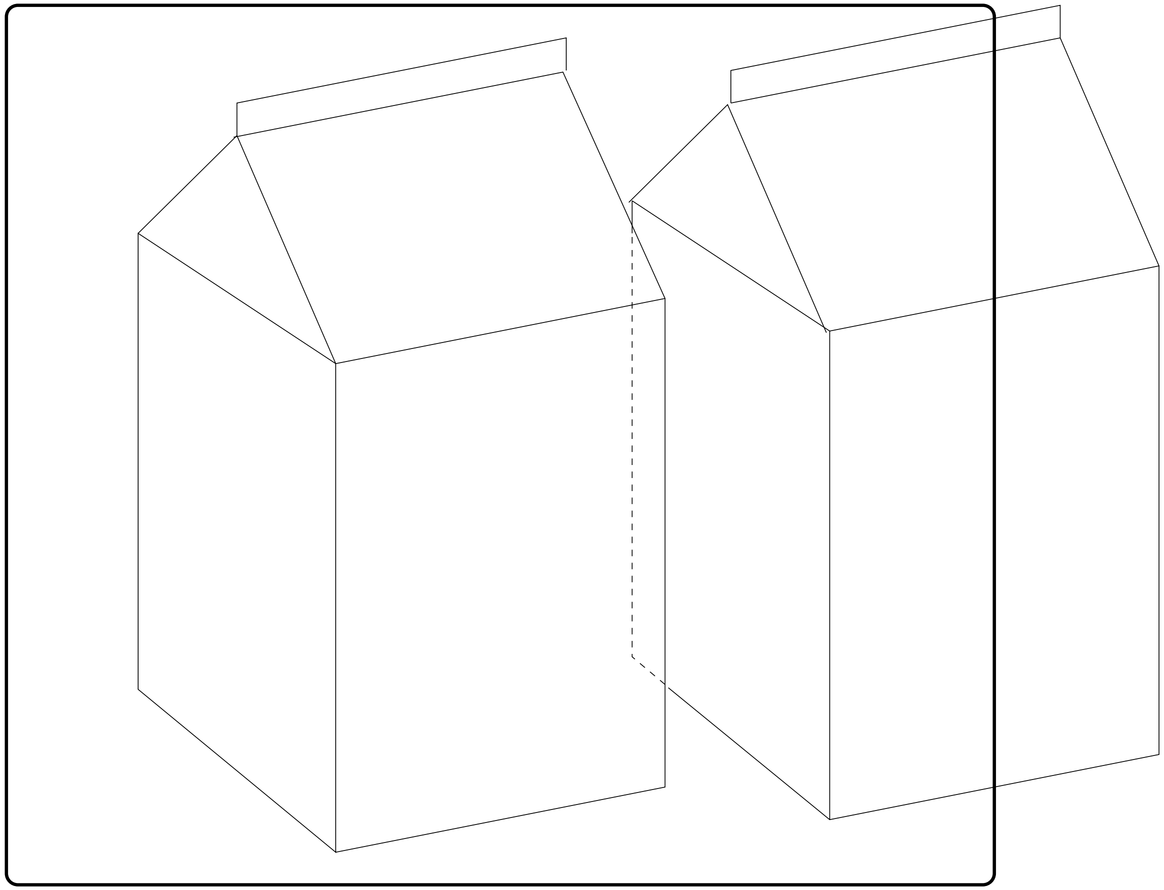
After transformation

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.

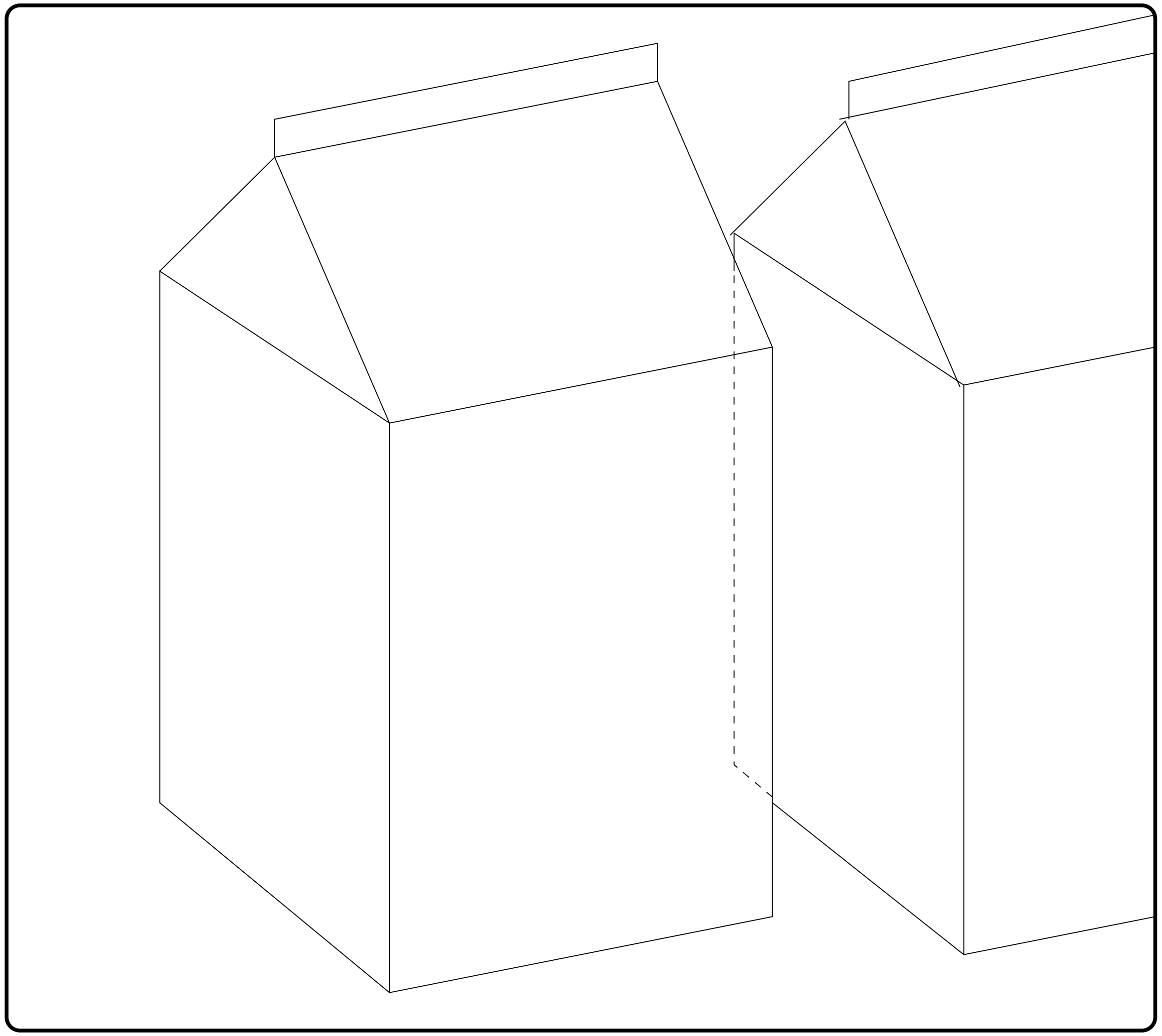


After hidden surface elimination

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.

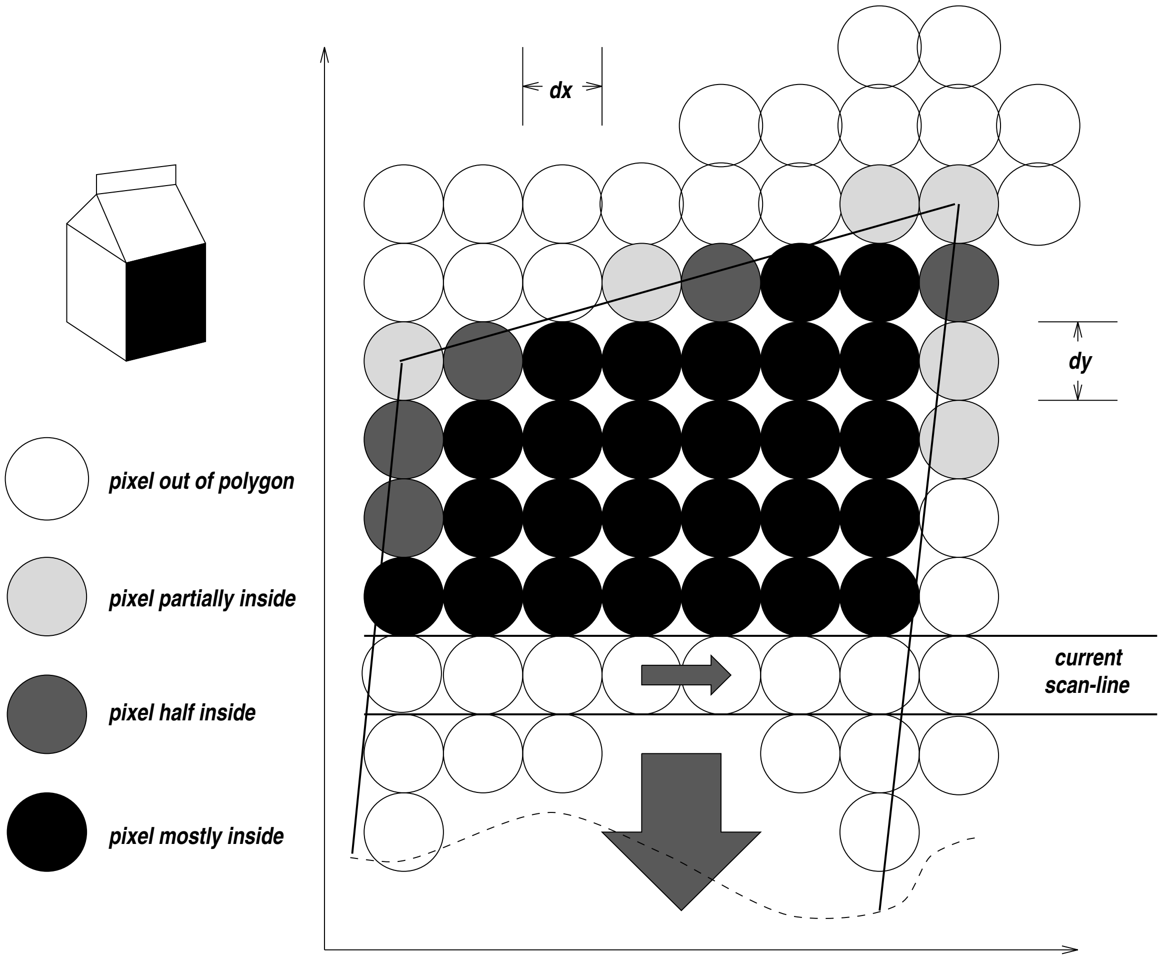
Rasterization

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.

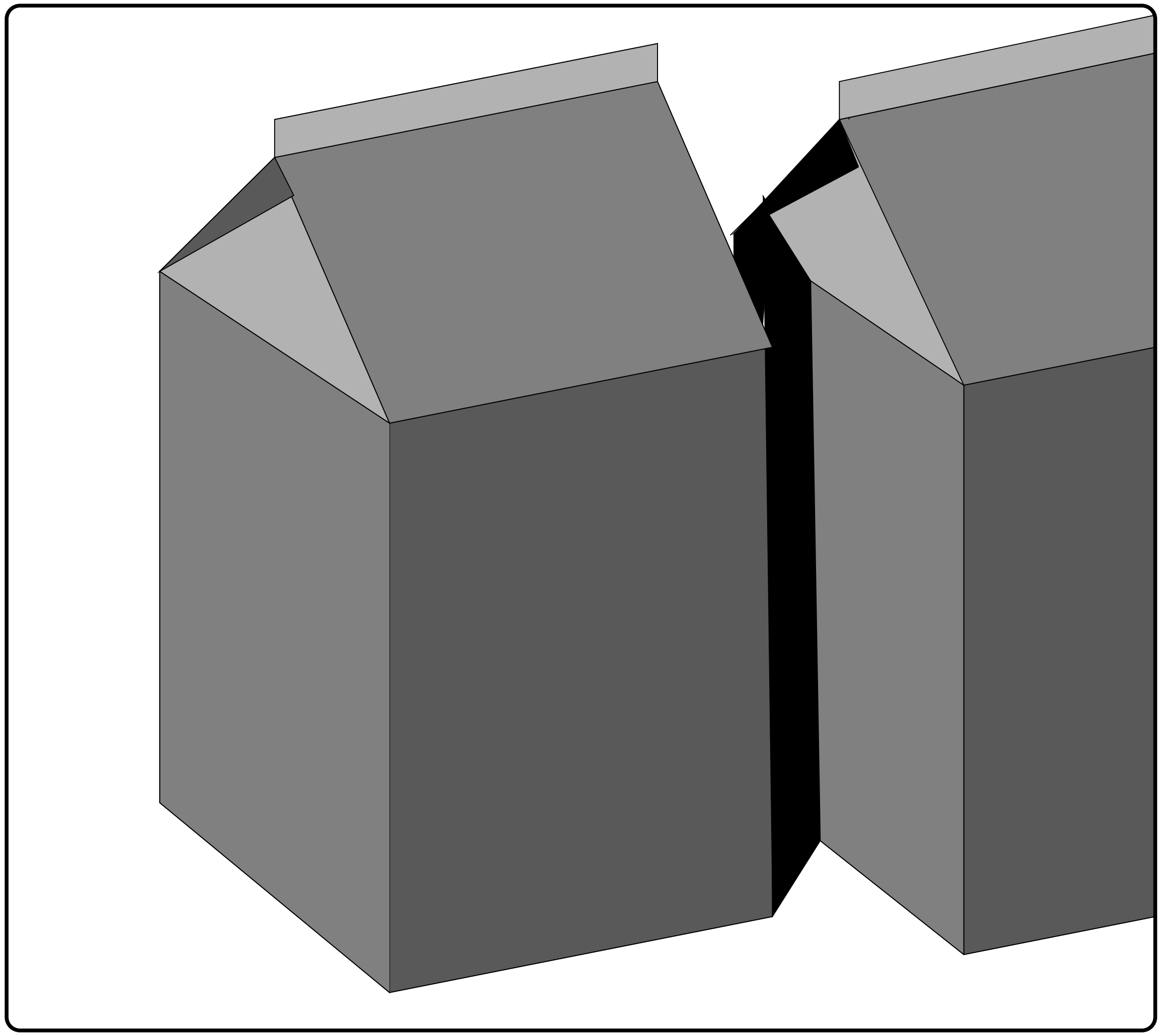
For a polygon based object, a *scanline* algorithm is commonly used to render the final image. The pixels are scanned a line at a time and their color is determined from the polygons that contribute to that part of the screen.

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.

TERRAIN RENDERING

Rendering of Terrain

Terrain rendering is a special case of 3-D volume rendering:

- The objects are static – only the viewer moves.
- There is only a single simply connected object – the land surface.
- A terrain model is basically a 2-D surface with a variable elevation in the third dimension, which is much simpler than a general 3-D problem. Terrain rendering is often called a 2.5 dimensional problem.

For these reasons, there are a number of optimizations and simplifications that can be made for a terrain rendering program that cannot be done for a general purpose volume rendering program.

Portability and Scalable Rendering

For best results, the rendering must be done on a high-performance computer. However the renderer should be portable, so it will run on parallel computers with different numbers of nodes, high-end graphics workstations, and even low-end platforms like Macs.

Our strategy is to make a scalable algorithm that will always run in near real time, by (gracefully) sacrificing image quality when there is less compute power available. This can be done relatively easily using polygon-based rendering by coarsening, i.e. reducing the number of polygons used.

This method requires handling two key problem areas which are interesting research topics:

- Keeping a constant frame rate for different computers and different scenes
- Creating optimal coarsenings of the polygons

Terrain Object Representation

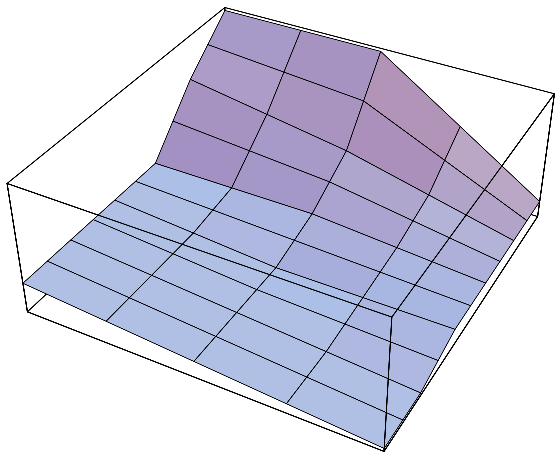
A terrain object is defined by an elevation (height) field and an image. The elevation field encodes information as elevation y for a specific (x, z) coordinate. The image gives the color associated with the coordinate. This information can be obtained from a digital elevation model (DEM) and a satellite image of the terrain.

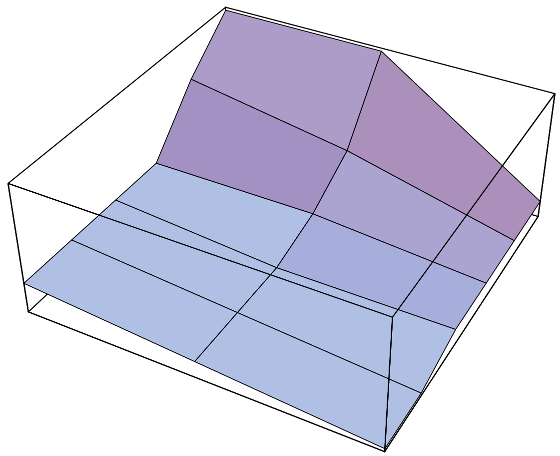
A terrain is a simply connected surface (i.e. no disjoint sections), so terrain rendering is a special case of volume rendering, sometimes described as 2.5-D rather than 3-D. This allows some optimizations to be used in the rendering process.

The terrain data set is stored as a number of two dimensional grids. Each grid contains a subsection of the terrain data set. The data is stored as triangular polygons made by triangulating the square mesh used as the basis of the DEM.

Polygon Representation

The terrain data can be stored at different levels of resolution. Coarser resolution polygons can be used for terrain further from the viewer. This will greatly reduce the computational requirement. Can do this very simply by just using a coarser mesh, but may be more effective to use an adaptive scheme with a very coarse mesh for flat regions (lakes, plains) and a less coarse mesh for mountains.





Shading and Shadowing

Because the light source remains constant and terrain is virtually non-reflective, shading and cast shadows do not change from one view to another. Thus a significant portion of the rendering process, the computation of shadows and shading, can be done in a preprocessing step.

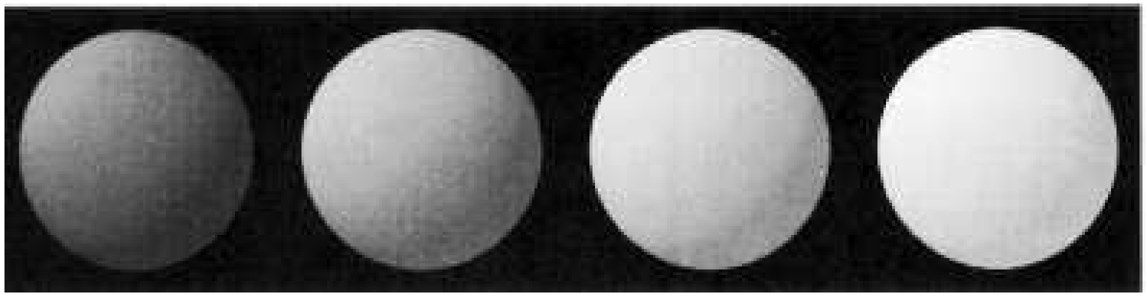
Shading can be modeled as two separate processes:

Diffuse light is uniformly reflected from surfaces, i.e. does not depend on the angle of the surface to the viewer, only to the light source.
Ambient light is radiated from all the objects in the scene, and is independent of the light source.

Diffuse light is blocked in areas of shadow, so must compute the areas in shade (occluded from the light source). Ambient light is not affected by shadowing.

Shading Model

Different materials have different parameters for ambient and diffuse reflection, which are used in a standard Lambertian shading model to calculate the intensities for ambient and diffuse light.

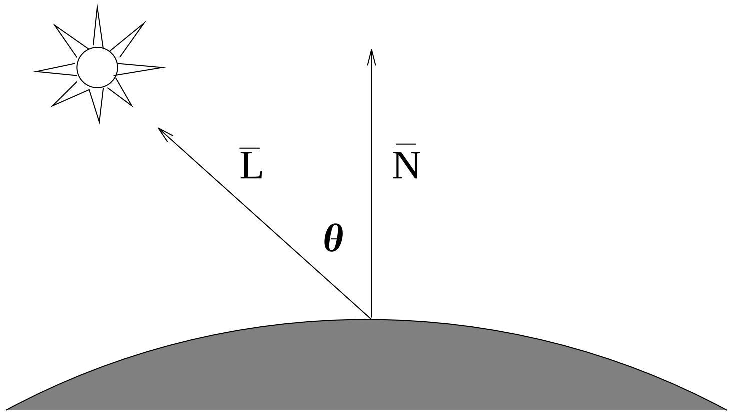


Example of a shaded terrain.

Shading Model

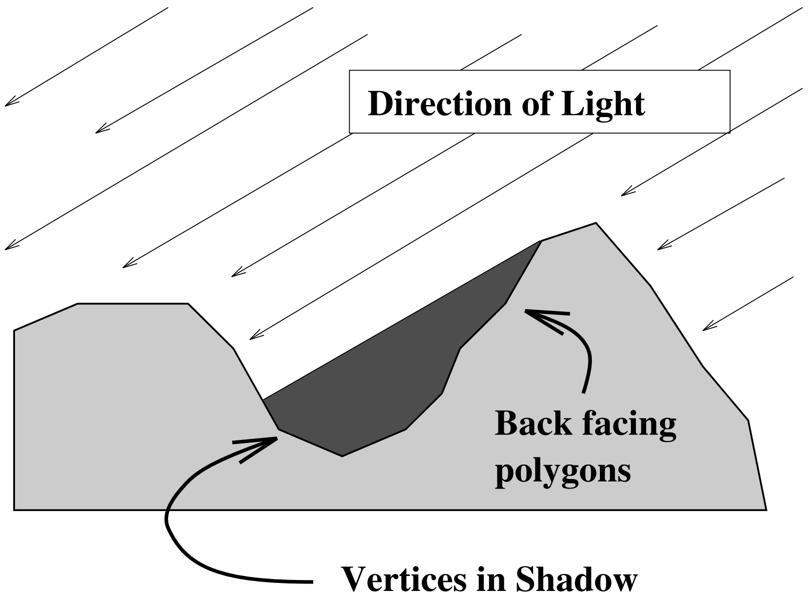
Ambient light	$I = I_a k_a$
Diffuse light	$I = I_p k_d \cos\theta = I_p k_d \overline{N} \cdot \overline{L}$
Together	$I = I_a k_a + S_i I_p k_d \overline{N} \cdot \overline{L}$

$$
\begin{aligned}
I &= \text{total intensity} \\
I_p(I_a) &= \text{diffuse (ambient) intensity} \\
k_a &= \text{ambient reflection coefficient} \\
k_d &= \text{diffuse reflection coefficient} \\
\overline{N} &= \text{normal to the surface} \\
\overline{L} &= \text{position of the sun} \\
\theta &= \text{angle between sun and normal} \\
S_i &= 0 \quad \text{if in shadow} \\
&= 1 \quad \text{if not in shadow}
\end{aligned}
$$



Shadowing

A Z-buffer algorithm is used to calculate the area in shadow, which is identified by finding the back-facing polygons.



REMOTE DISPLAY OF RESULTS

Client/Server Model

Since real-time rendering is so compute-intensive, it must be done on a high-performance computer to get the best results. However we want to be able to display the results on a workstation on the user's desk, or even on a Mac or PC in a high school.

The results must therefore be sent interactively across a high-speed network from the rendering engine to the display monitor. The rendering engine will be running on a high-performance computer, or perhaps a high-end workstation with special graphics hardware.

We must therefore deal with issues of networking and image compression.

Networking Issues

Require about 30 frames per second for realistic animation. A typical frame size would be 640×480 pixels, and ideally each pixel would have 24 bits of color information. This would require about 220 Mbits per second of bandwidth to send the raw image.

Current FDDI and ATM networking technologies available at NPAC can achieve about 100 Megabits per second, however actual performance (using software such as PVM) can be an order of magnitude less than this. This is clearly a difficult but crucial problem that needs to be addressed.

There are three ways of reducing the bandwidth required:

- Reduce the image size (not a favorable option)
- Reduce the frame rate (10-20 fps is probably adequate)
- Reduce the image size by compressing the image

Image Compression

Images need a lot of data, and animated images are even more of a bandwidth hog, so there is a lot of interest in methods for compressing images and video. JPEG compression is fast becoming the industry standard for images, MPEG for video. Other methods include Wavelet and Fractal compression.

JPEG decreases image quality (i.e. is lossy), but in areas that are insensitive to human vision. At compression ratios of 10:1 the image is virtually indistinguishable from the original, and generally can achieve 20:1 without serious loss of image quality.

However standard JPEG compression algorithms are much too slow for real time animation (about 0.2 seconds to compress a 320×200 image on a DEC Alpha). Fast parallel or hardware implementations are needed. Decompression is fast, and cheap special purpose chips will soon become standard features.