Computer Graphics

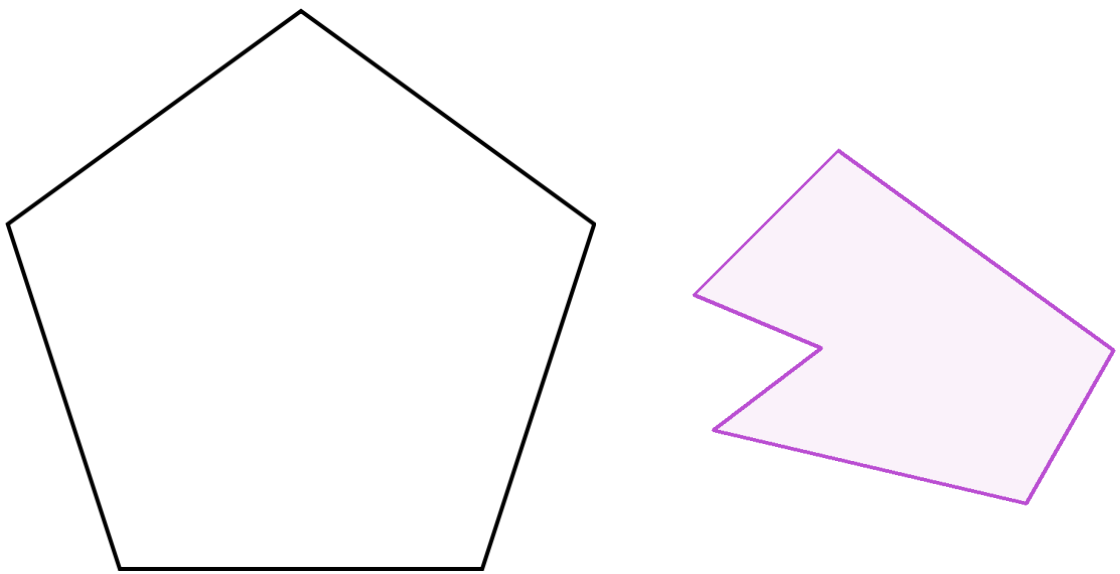
Definitions

- **Viewpoint** - where the viewer (camera) is located and it's orientation.
- **Viewing frustum** (viewing volume) - the areas visible from the viewpoint. When using perspective it is frustum shaped (four sided pyramid with the top removed). Only objects in this volume are visible and need to be rendered.
- **Frame buffer** - the memory where an image is stored which will be displayed on the screen.

Definitions

- **Convex** - a shape which does not have any cuts in it. More formal definitions of convex are:
  - Each interior angle is at most 180 degrees (always turn in the same direction if you follow the perimeter).
  - If you join any two points on the shape, the line created will be entirely inside the shape.

  Objects which are not convex are called **concave**.



Convex and concave polygons.

Shading Models

- How to shade flat polygons so they appear as smooth.
- Current methods use the **Phong reflectance model**. This model deals only with light reflected from light sources and not from other objects. It uses three components to illuminate surfaces, these are ambient, diffuse, and specular reflections.
- Gouraud Shading, Phong Shading, and Blinn-Phong shading are all applications of the Phong reflectance model. They differ in which points on the surface they calculate, how they calculate those points, and what is interpolated.

Phong Reflectance Model

- The Phong reflection model is:

$$I_p = k_a i_a + \sum_{lights} (k_d (L \cdot N) i_d + k_s (R \cdot V)^{\alpha} i_s)$$

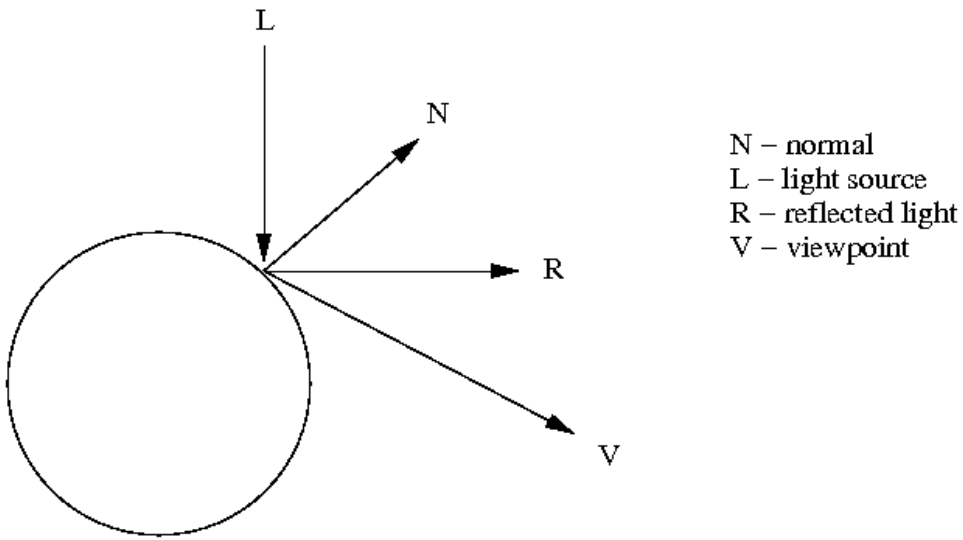
  where L, N, R and V are vectors pointing towards the light, normal, reflected light, and viewpoint respectively. The i_a, i_d and i_s values are intensities of the ambient, diffuse, and specular lights. Note that i_d and i_s are different for each light source but i_a is constant for the scene. The k_a, k_d and k_s values represent the ambient, diffuse, and specular reflective properties of the material. The α value is the specular reflection exponent which controls how well defined the specular highlights will appear.
- This equation is normally used once for each rendering colour (RGB).

Phong Reflectance Model

- The Phong reflectance equation can be read as:
  - for each surface, calculate how bright it will be lit by adding the amount of ambient, diffuse and specular light striking the surface for each light source (controlled by the summation $\sum$ and the additions)
  - each surface can reflect each type of light differently (through the k values)
  - the brightness of the ambient light is the same in the whole scene (although different objects can reflect it differently at different levels through the k values)
  - diffuse light is affected by the position of the light (L) and the normal (N)
  - specular light is affected by the position of the reflected light (R) and the viewer (V)
  - the distinctness of specular reflections is controlled by α (the higher the value, the smaller and sharper the specular reflection will be)

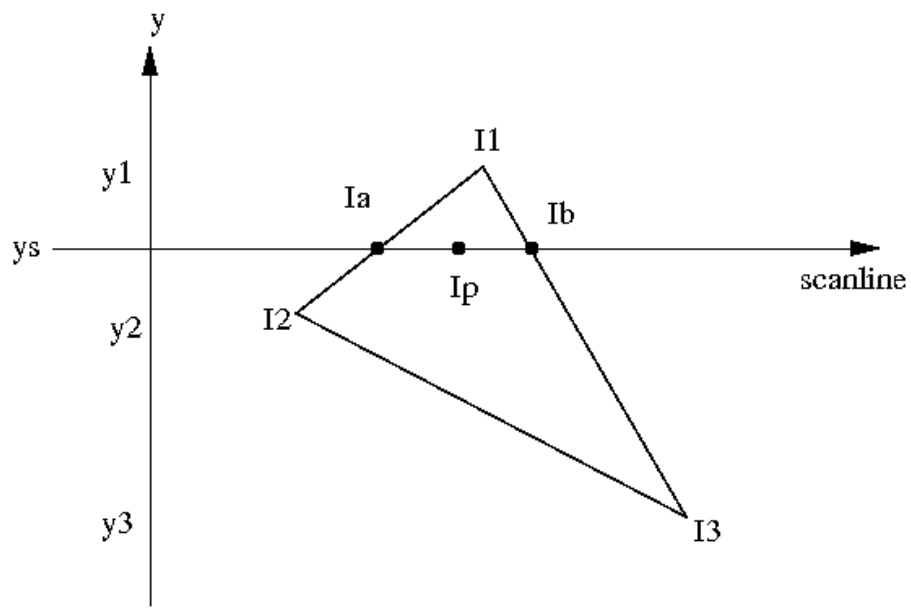
Phong Reflectance Model

- The vectors used in the Phong reflection model.
- As the viewpoint V moves closer the the reflected light R the specular light increases.
- The largest amount of reflected light is at an angle equal the angle between L and N.



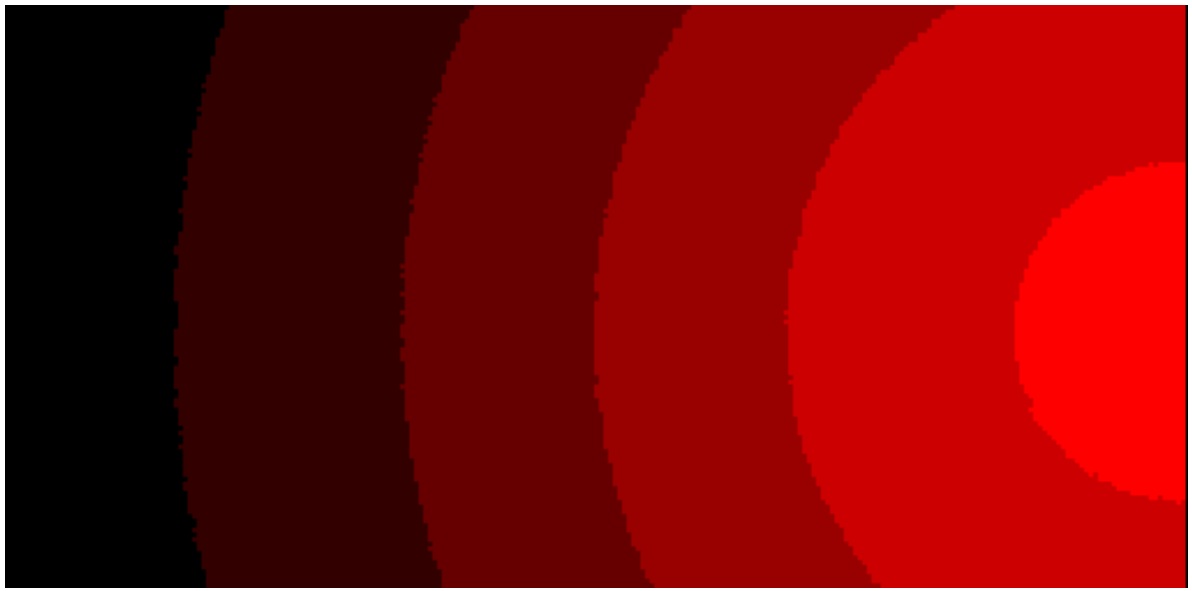
Gouraud Shading

- Uses the Phong reflectance model at each vertex to calculate the intensity and then interpolates them across the polygon.
- Interpolation is linear.
- $I_a = I_1 - (I_1 - I_2)\frac{y_1 - y_s}{y_1 - y_2}$
- $I_b = I_1 - (I_1 - I_3)\frac{y_1 - y_s}{y_1 - y_3}$
- $I_p = I_b - (I_b - I_a)\frac{x_b - x_p}{x_b - x_a}$



Gouraud Shading

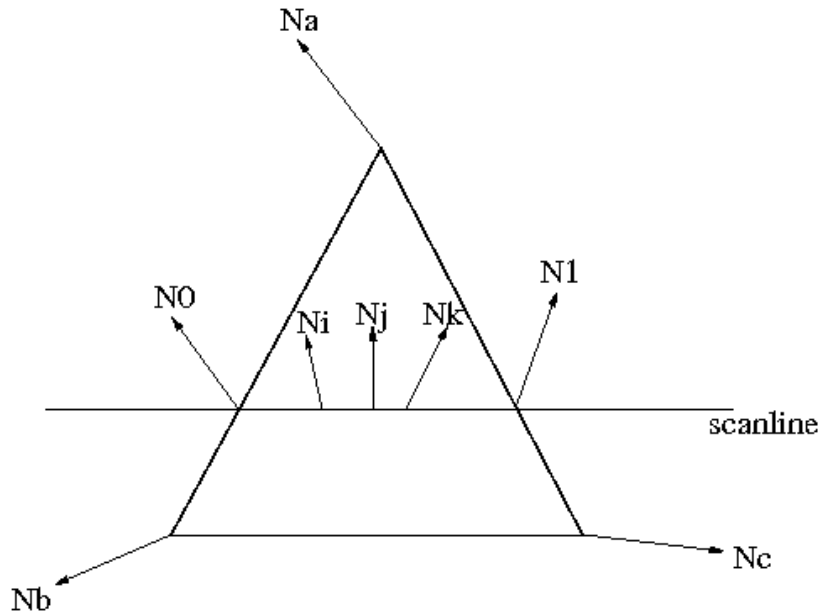
- Also called intensity interpolation shading or colour interpolation shading.
- Requires **vertex normals** to calculate **vertex intensities**.
- Rapid changes in intensity can produce colour banding.
- Linear interpolation is easy to implement in hardware.



Colour banding.

Phong Shading

- Also called normal-vector interpolation shading.
- Interpolates normal vectors across the polygon. The normals are used to calculate intensity.



Interpolating normals for Phong shading. Normals N_a, N_b and N_c are calculated, all others are interpolated.

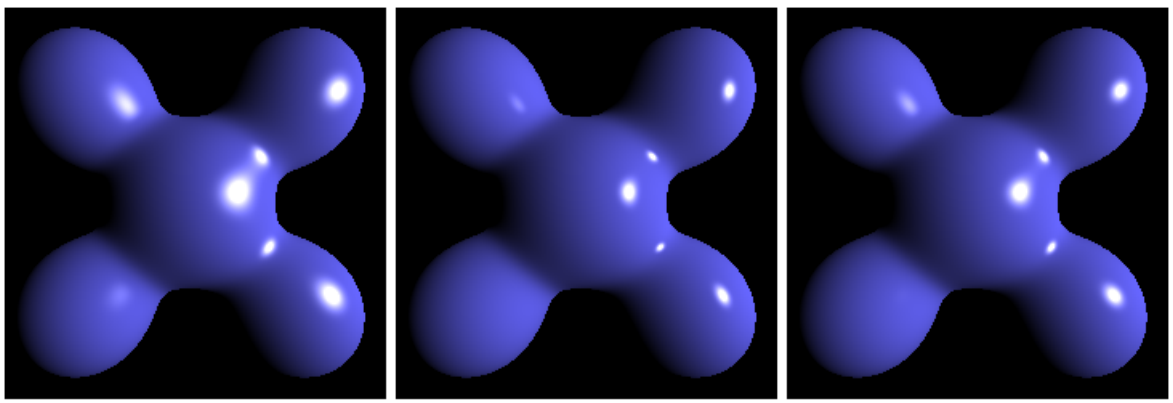
Phong Shading

- Produces better results than Gouraud shading. Specular highlights are more accurate and less likely to move.
- Requires lighting calculations per pixel pixel instead of per vertex. Gouraud only requires interpolation per vertex.
- Specular reflections occur when the reflected light is directed towards the viewpoint (when the R and V vectors are similar for light which is reflected through the normal). Interpolating normals allows the reflection to be calculated anywhere on the surface instead of only at the vertices. Gouraud shading can miss the highlight if it doesn't fall on a vertex.

GouraudSample demo

Blinn-Phong Shading

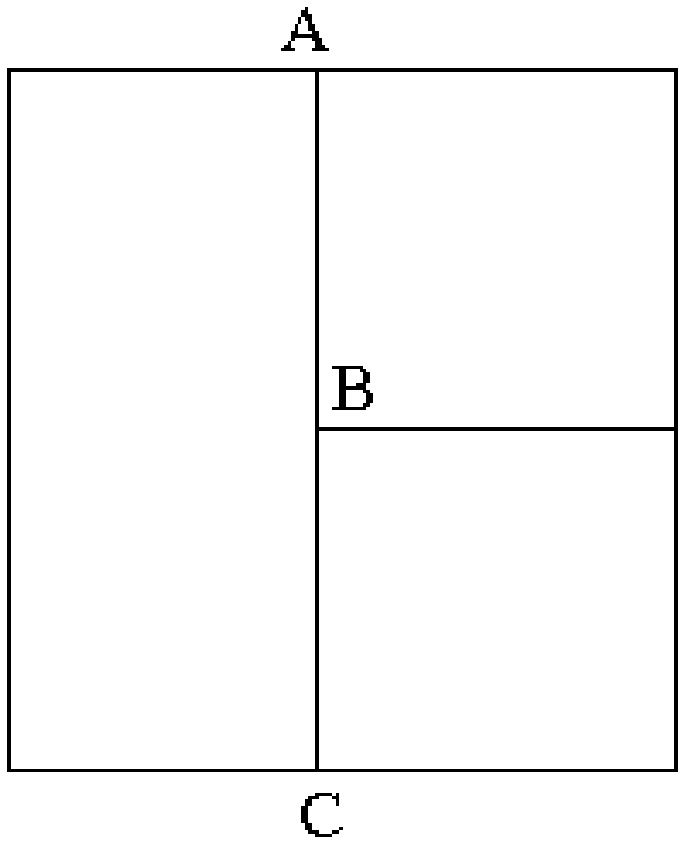
- It is also called the Blinn-Phong reflection model or the modified Phong reflection model.
- It trades visual precision for computing efficiency.
- It replaces the $R \cdot V$ calculation with $N \cdot H$ where $H = \frac{L+V}{|L+V|}$.
- H is the halfway vector between the viewer and the light source.
- The value of $N \cdot H$ is lower than $R \cdot V$ but this can be adjusted using α.



Problems With Interpolated Shading

- The object's silhouette will not appear smooth but the shading may make the surface of the object appear smooth.
- Interpolations are calculated without considering perspective. Perspective can move vertices but shading is calculated for untransformed points.
- Shading can change based on the orientation of a polygon (orientation dependence).
- Adjacent polygons which don't share a vertex may not be shaded similarly.

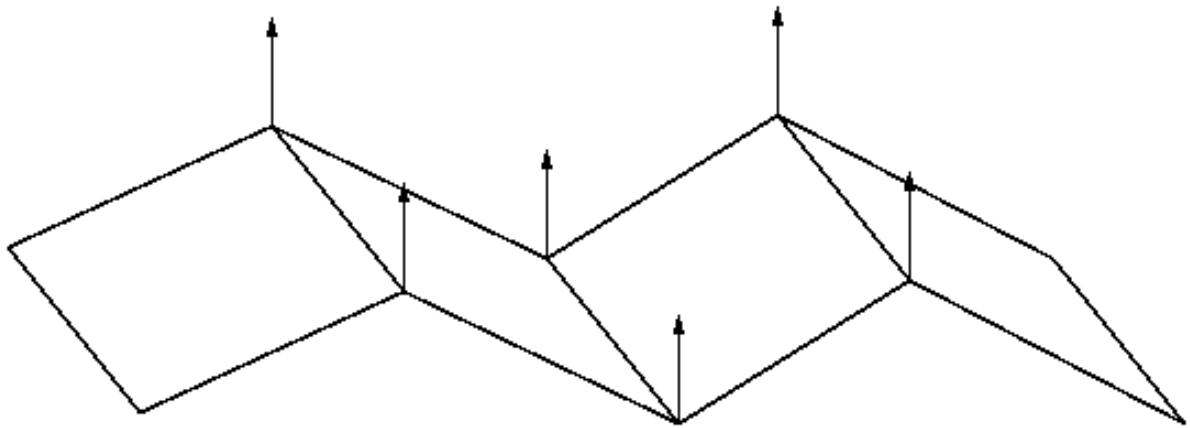
Problems With Interpolated Shading



There may be a shading discontinuity at vertex B because the polygons on either side of AC are interpolated differently. If a specular reflection occurs at vertex B then it will only be visible on the right side of the surface.

Problems With Interpolated Shading

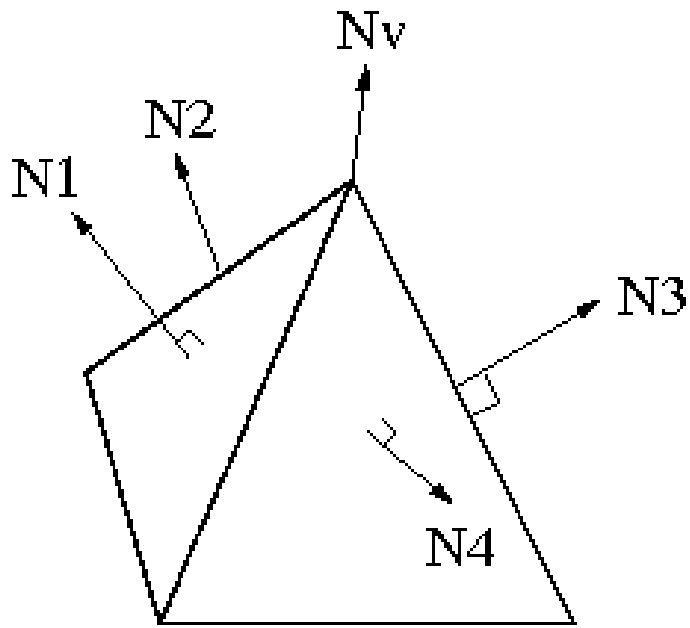
- Normals may not correctly represent the shape of the surface.



All normals point in the same direction. All of the surfaces will be shaded similarly even though they point in different directions.

Calculating Vertex Normals

- Vertex normals can be approximated from **surface normals**.
- Surface normals can be calculated by taking the cross product of three points on the surface.
- The approximate vertex normal is the average of all of the surface normals for polygons adjacent to the vertex.



The vertex normal N_v is calculated using $N_v = \frac{\sum_i N_i}{|\sum_i N_i|}$.

Visible Surface Determination

- Also named hidden surface removal.
- Identifies the parts of a scene which are visible from a given viewpoint.
- Improves performance by avoiding the drawing of objects which are not visible.
- Most methods involve either sorting the polygons from back to front or group them into regions.
- **Back face detection** is a method to cull out all polygons on the non-visible (back) of an object. Approximately half of the polygons in an object can usually be culled using this method. It requires vertices to be ordered either clockwise or counterclockwise so the polygons on the back can be easily identified. Commonly used in all graphics systems.

Back Face Culling

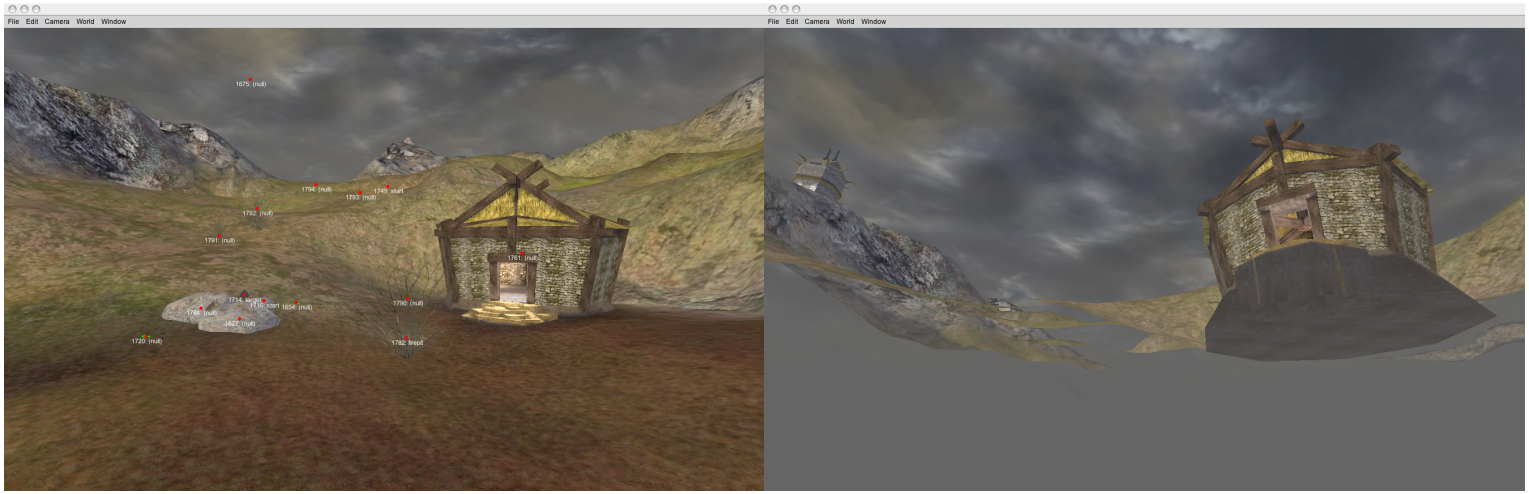
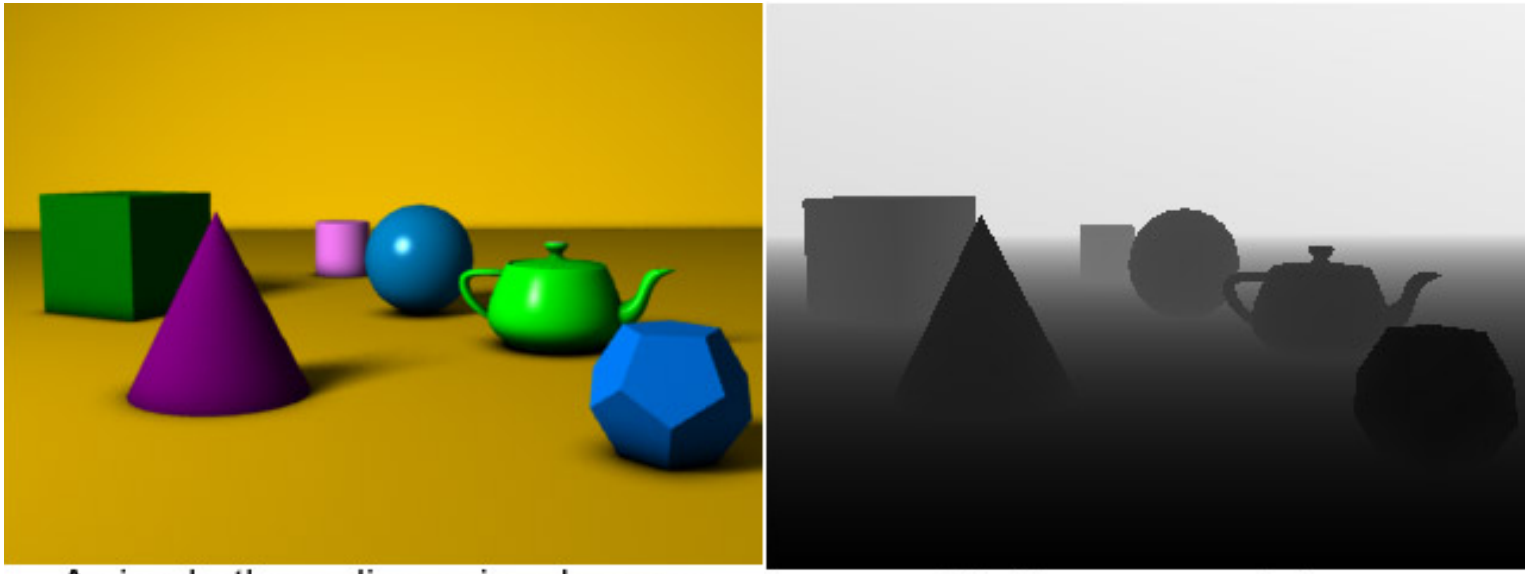


Image on the right shows back face culling of ground when viewpoint is below the surface.

Depth Buffering

- Also called Z-buffering.
- Requires a memory buffer that stores the distance from the viewpoint for each pixel drawn. When an object is drawn, each pixel for the object is tested against the Z-buffer to determine if the new object is in front or behind previously drawn objects. The pixels which are in front of previously drawn objects are drawn in the framebuffer. Those which are behind previously drawn objects are not drawn.
- The algorithm doesn't require sorting the polygons before drawing them with the exception of transparent surfaces. These must be sorted and drawn from back to front after all of the opaque objects have been drawn. This must be done so the transparent colours blend properly with the objects behind them.
- Commonly used in all modern games. Supported by hardware.

Depth Buffering



A simple three dimensional scene

Z-buffer representation

Sample of an image and its associated Z-buffer. The lighter shades in the Z-buffer indicate greater depth.

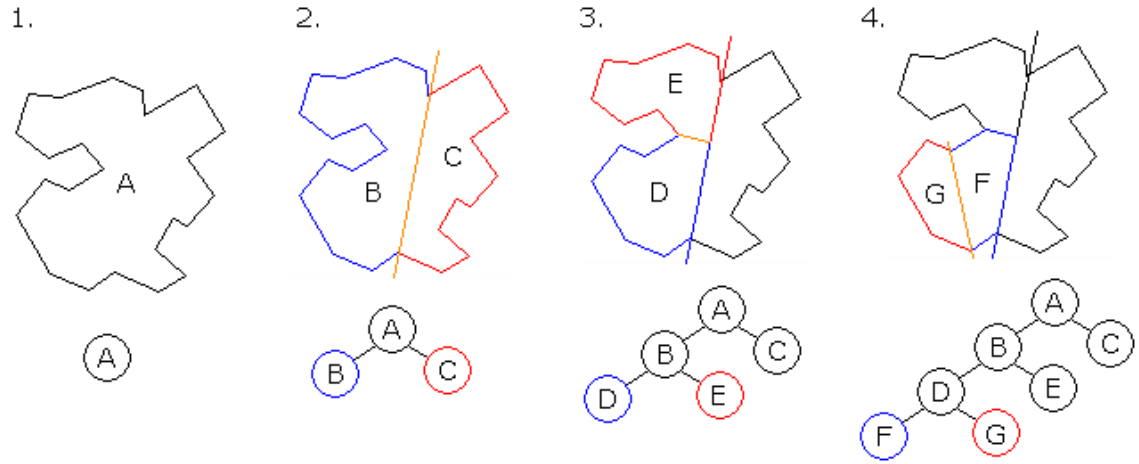
Binary Space Partitioning (BSP Trees)

- Implements the **painter's algorithm** which means objects in the scene are drawn from back to front. Objects which are far away will be overdrawn by nearer objects. This can be wasteful if a lot of objects are overdrawn.
- One traversal of the tree will order the polygons in the scene from any viewpoint. Traversed in linear time.
- Recursively divides space into non-overlapping sets and stores them in a tree structure. Divides space into convex subspaces.
- Works best with static scenes, for structures such as buildings where only the viewpoint changes. This is suitable for things such as flight simulators, scenes where the buildings don't change but objects and players can move.
- The tree is pre-computed. It is loaded during gameplay.
- Used to render the scene quickly and for collision detection.

Binary Space Partitioning (BSP Trees)

- It's possible to determine which parts of the scene are outside the viewing volume.
- Each subspace is divided into two parts and stored in a binary tree.
- Dividing planes are created by extending polygons in the scene. It's important to choose appropriate dividing polygon (plane) or the tree can become unbalanced which leads to less efficient traversal.
- Dividing planes are often chosen arbitrarily. If a poor choice is made then other options are often tested.
- Can be combined with other algorithms such as the Z-buffer.
- Used in the Doom engine.

Binary Space Partitioning (BSP Trees)



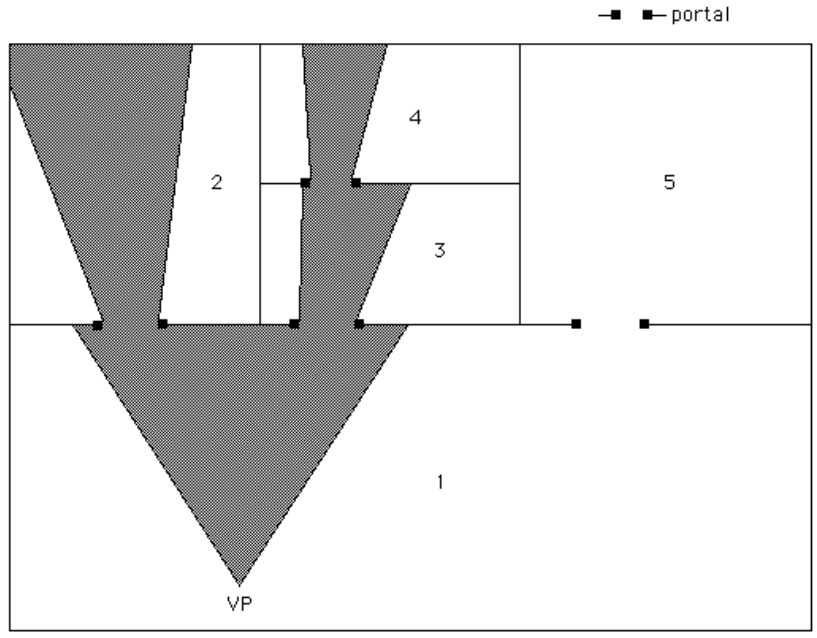
Building a BSP tree from a 2D region.

1. A is the root of the tree and the entire polygon.
2. A is split into B and C.
3. B is split into D and E.
4. D is split into F and G, which are convex and hence become leaves on the tree.

Portals

- Useful to divide up static spaces so only visible sections are considered for rendering (visible to the viewpoint).
- Space is partitioned into sets (sectors) which are visible at one time. Adjacent sectors are linked by portals. These are normally doors and windows.
- If a portal is visible then the system must test to see what part of the adjacent sector is visible. If a portal to a sector isn't visible then that sector will not need to be rendered.
- They are most useful for enclosed spaces such as inside buildings or in tunnels. Open spaces generally don't have the small openings to server as portals.
- Can be used to create effects such as mirrors and doorways to other locations by attaching non-adjacent sectors to a portal.

Portals



Room with adjacent portals. Viewpoint is in room 1. Room 2 can be seen through the left portal. Room 3 can be seen through the middle portal and room 4 can be seen from the portal in room 3. Adding rooms (sectors) stops when the viewing volume doesn't intersect with any portals. Room 5 is never in the viewing volume and isn't drawn.

References

Many images are taken from:

- Torque Game Engine 1.5 Demo, www.garagegames.com.
- Wikipedia web site, www.wikipedia.org.