Game Graphics

305900
Fall 2009
10/26/2009
Kyoung Shin Park

Isometric Games

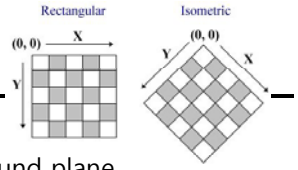
Sprite-based graphics

- In the past, the way of doing game graphics
- Still commonly used
  - Mobile phones
  - Internet games
  - Casual games
  - Serious games
- Advantages
  - No graphics hardware required
  - High quality, also on low resolution
  - Much detail
  - Relatively easy to use and control
- Problems
  - Animation speed difficult to control
  - Lost of memory required
  - Fixed viewpoint

Sprite-based graphics

- In general, sprite games are
  - Top down (e.g. maze games)
  - Front facing (e.g. platform games)
- Clipping for showing a smaller part (views/scrolling)
- Hidden surface removal
  - Drawn from back to front
  - Transparent area around sprites
- Overlay state information on top of background graphics or use separate panel to show state information
- Depth can be achieved using layers
  - Use parallax scrolling
  - Use a different levels of Sprites' size
- Collision detection
  - Use bounding boxes
  - Pixel-based collision detection
    - Cheap enough by first doing bounding box test
    - Low performance and slow. Hence, this method is used only when action is required.

Basics of Isometric Games



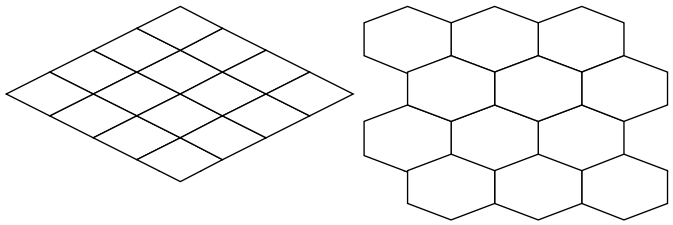
- Isometric game
  - Each object has a position (x,y) on the ground plane.
  - Possibly, each object has a height (z) above the ground plane.
  - The game world is displayed in an isometric projection.
    - Parallel projection
    - 45 degree angles with three major axes
    - Advantages: Good overview & View all objects in the same size.
    - Disadvantages: Perspective distortion, especially in orthogonal axes
    - Can be improved by limiting the view
    - Distortion along vertical axes is the worst
  - Most operations are done in the ground plane rather than in 3D.
  - Hidden surface removal by depth sorting
    - Because it's easy to implement
  - Use it in many games
    - Command and Conquer (except Generals) , Diablo, Age of Empires , SimCity, etc

Sprites

- In computer graphics or games, a sprite is a 2D image or animation that is integrated into a larger scene.
- Originally invented as a method of quickly compositing several images together in 2D video games.
- In general, 2D game figures are all referred to as sprites.

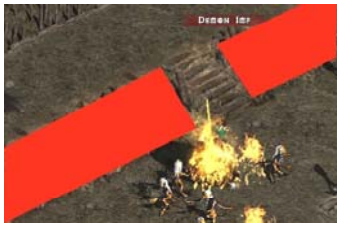
Tiles

- In traditional video or computer games, a tile-based game means a game which uses tiles as part of its graphic output and/or unit movement system.
  - Tiles are small rectangular, square, or hexagonal graphics images.
  - Tile-based game usually simulate a top-down or isometric view on the playing area and are almost always 2D.
  - Fixed size (because no perspective) so equal at all depths.



Tiles

- Tiles are need to be carefully aligned in order to hide the underlying tile structure.
  - A little bit of variations (especially important, if adjacent tiles have similar terrains)
  - Height differences are often only visual (but, height are not actually implemented in the game)
  - Non-transparent, static objects



Tiles

- Make tiles as 3D look-and-feel that the major axis are clearly visible
  - Streets, walls, etc (in graphics)
  - Well-known but weird key mapping (in motion)
- Can be composed in advance, or on the fly
  - Needs tile alignment rules
  - Usually, created from a higher-level representations – road, river, mountain, ocean, etc.

Age of Wonder (2003),
a dimetric hexagonal tile-based game

Structures on the Surface

- Static, fixed objects on the ground - trees, buildings, etc
  - Represented by fixed position on the plane (2D)
  - Often aligned with grid
  - Transparent sprites
  - Can be animated
  - These can either be objects in the game or drawn on the background or foreground (if not animated)
- Moving units on the ground - people, vehicles, etc
  - Represented by their position on the plane (2D)
  - Do not need to stay on the tiles
  - Can move behind structures and other units
- Moving units on top of ground - bullets, balloons, etc
  - Represented by point on plane with a height
- Movement
  - User interface (E.g., arrow key to move direction)
  - Preferred diagonal motion
  - Different horizontal and vertical speed

Collision Detection

- Avoids complicated 3D calculations
- Sprites can be overlapped while objects can't.
- Stores the object's shadow on the ground plane
  - Or some approximation or relevant part of the shadow
  - If required, stores its height interval
- Assumes ground projection is convex, otherwise split in pieces
- For collision detection
  - Checks whether shadows intersect
  - If so, checks whether their height intervals intersect
  - If so, a collision is detected
  - But, sometimes more complicated checking is required (shape dependent)

Collision Detection

- When there are different height intervals at different places, also split
  - E.g., If a gate consists of left, middle, right pieces, you might not store a mask at all for middle pieces (no collision is occurred when only people are walking around)
- In game design,
  - Uses a mask for the object that is the shadow of the sprites.
  - Collision is based on the masks.
  - Specifies height intervals if required.

Hidden Surface Removal

- Drawing order for hidden surface removal
  - Backgrounds
  - Objects
  - Foregrounds
  - Overlays
- Each object has a position on the ground plane.
  - Position has to be carefully chosen. – e.g. in the center of the shadow
- Drawing order for isometric view
  - Sort the positions from top to bottom
  - Then, far objects are drawn first

Hidden Surface Removal

- However, drawing order is not always completely correct, in particular when the positions are close to each other.
- When shadows overlap and there is a height,
  - Sort by increasing height.
  - Then, draw in this order, so objects that are below other objects are drawn first
- In game design,
  - Select the origin of the sprites at the ground position (can be outside sprites)
  - In each step, set depth=-y (to guarantee the depth sorting)
  - If there is a height interval, adapt the depth in a more involved way, taking the interval in account.

3D Graphic Games

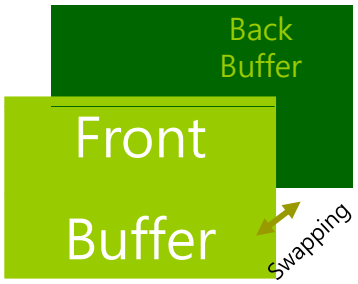
3D Graphics

- Frame buffer and Double buffering
- Visibility and Depth buffer
- Alpha blending, Stencil buffer
- Coordinate Systems
- Triangle, Vertex, Index, Polygon, Mesh
- Transformations
- Camera
- Lightings, Shading, Materials
- Textures
- Billboarding
- Normal mapping, Bump mapping, Displacement mapping

Frame Buffer

- Frame buffer is a chunk of graphics card memory that contains what is displayed on the screen. Like an image, but for each pixel that can be additional data besides color, depth, mask, etc.
- Color buffer
  - Double buffering – front buffer & back buffer
  - Stereo buffer – stereoscopic image
  - Alpha buffer - transparency
- Depth buffer – hidden surface removal
- Stencil buffer – blending
- Accumulation buffer – anti-aliasing, motion blurring

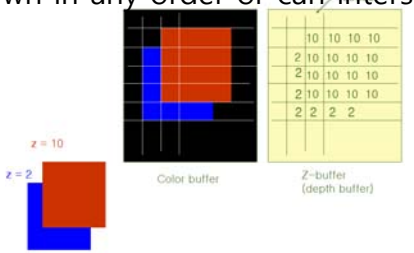
Double Buffering



- Image is drawn to the back buffer while the front buffer is displayed to the viewer.
- Swapping is usually done during vertical blank of the monitor (50Hz).
- Most PCs tend to turn this synch off resulting in flickering of graphics.
- Double buffering is useful for smooth animation

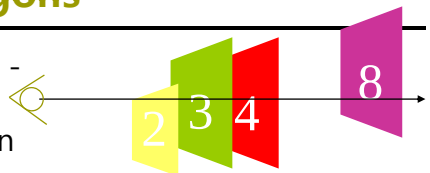
Visibility and Depth Buffer

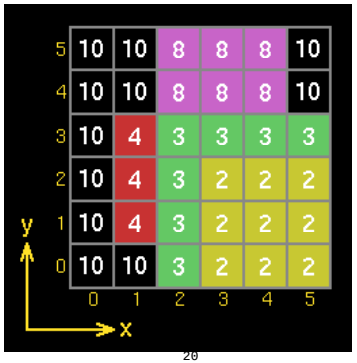
- In computer graphics, depth buffer has the same resolution as color buffer.
- Z-buffer(depth buffer) contains depth of each pixel drawn while color buffer contains RGB color of each pixel drawn.
- In Z-buffer algorithm, when drawing a new pixel, compare new depth to what's stored in z-buffer.
- Polygons can be drawn in any order or can intersect.



Depth Sorting of Polygons

- PS1 did not have Z Buffering - Polygons had to be sorted back-to-front to draw them in the right order.
- Z Buffer ~ 16-32bits/pixel.
- Buffer that stores depth information on a per pixel basis.
- Used to determine which pixel is in front of the other.
- Small Z values are in front.
- Avoids having to sort polygons in Z when rasterizing.

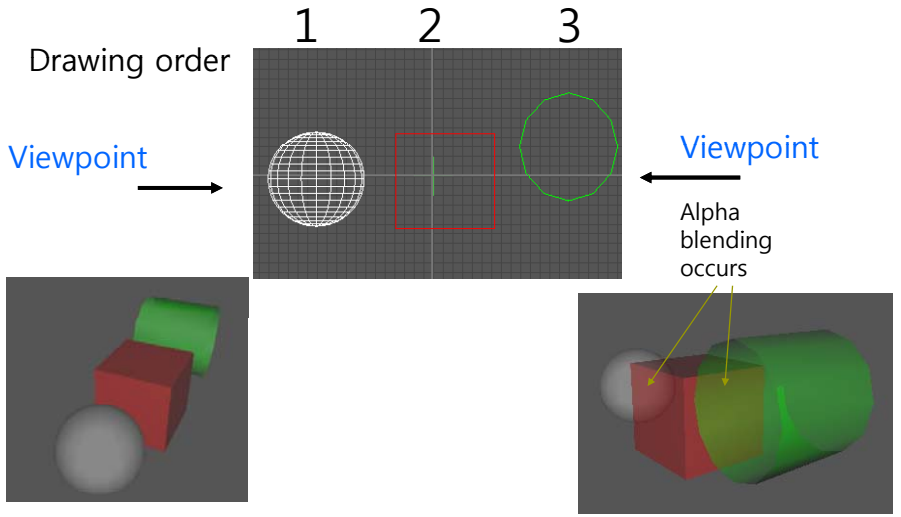




Blending and Drawing Order

- Blending
  - The alpha channel is the 4th color in RGBA
  - Opacity α=1: completely opaque & α=0: transparent
  - Color of the source pixel s=(s_R, s_G, s_B, s_A)
  - Color of the destination pixel d=(d_R, d_G, d_B, d_A)
  - Alpha blending = $S_A * S_{RGBA} + (1 - S_A) * D_{RGBA}$
- Blending functions are order dependent.
- If we draw a group of polygons that are some opaque and some transparent,
  - Draw opaque first follow by transparent polygons
  - Opaque polygons block all polygons behind them and affect the depth buffer
  - Transparent polygons should not affect depth buffer
- Draw back-to-front order if all are transparent polygons

Blending and Drawing Order

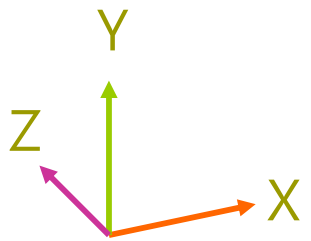


Stencil Buffer

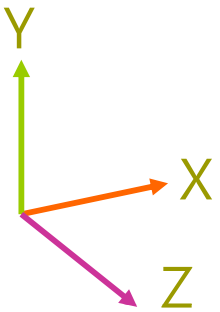
- The stencil buffer is like the depth and color buffers, but is an off-screen buffer for special effects.
- Using stencil buffer, particular portions of back buffer cannot be rendered – compare the stencil reference value with mask, to determine rendering of a specific pixel.
- Often used as masking or culling, e.g. mirror or shadow

Coordinate Systems

Left-handed Coordinate System (Typical computer graphics reference system – DirectX)



Right-handed Coordinate System (Conventional Cartesian reference system - OpenGL)

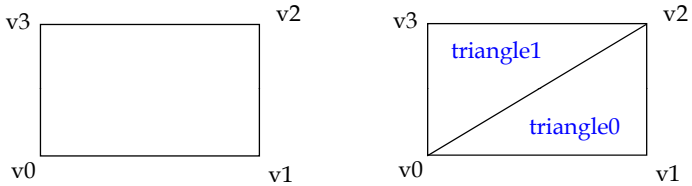


Vertex

- Vertex
- Additional characteristics
  - Surface normal
  - Color
  - Texture coordinate
  - And additional data for shader program..
- A triangle is composed of three vertices
  - Vertex is shared by adjacent triangles

Triangle

- Triangles are fundamental primitives for drawing 3D objects.
- Triangle has vertices, textures, colors.

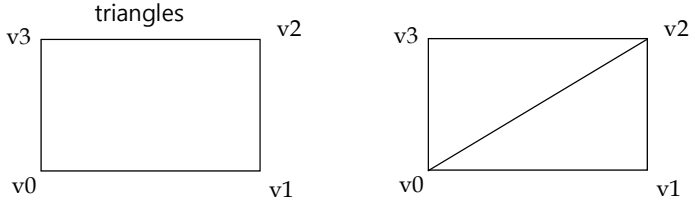



```
Vertex rect[6] = { v0, v1, v2,        // triangle 0
                   v0, v2, v3 };      // triangle 1
```

- Every triangle is defined by vertices in the correct winding order (depending on LHS or RHS)

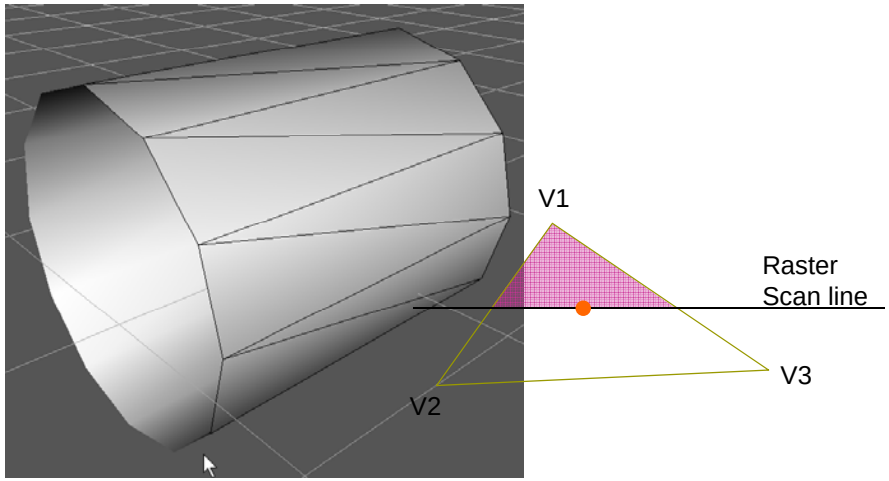
Index

- Index
  - If index list is not used, too many vertices are duplicated.
  - Use vertex list and index list to reduce duplicated vertices.
    - The vertex list contains all vertex information
    - The index list contains all indices of vertex list to represent triangles




```
Vertex vertexList[4] = { v0, v1, v2, v3};      // vertex list
int indexLIST[6] = {0, 1, 2,                   // index list
                    0, 2, 3 };
```

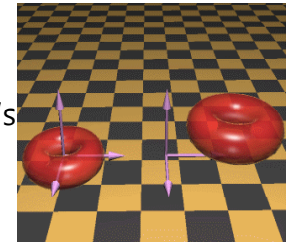

Polygons, Meshes



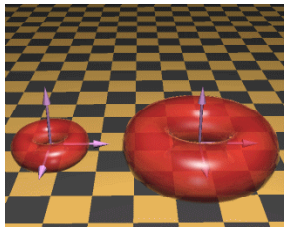
Transformations

- Transformation occurs about the origin of the coordinate system's axis

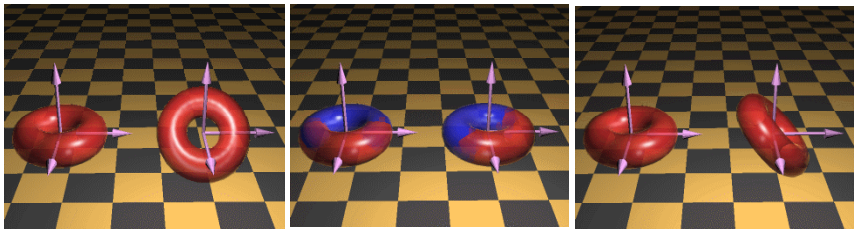
Translate



Scale

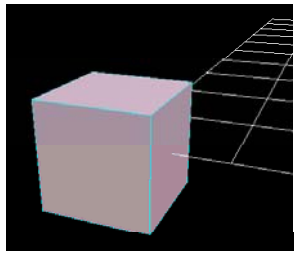


Rotate

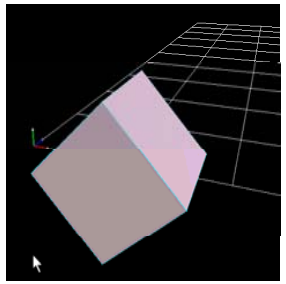


Transformation Order Matters

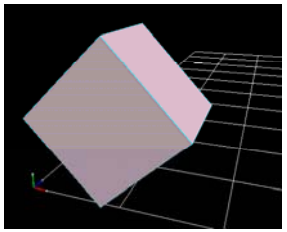
Box centered at origin



Rotate about Z 45; Translate along X 1

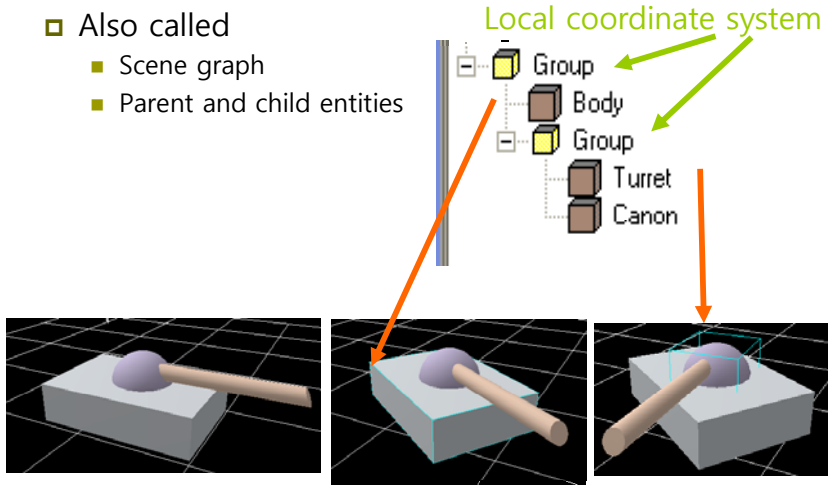


Translate along X 1; Rotate about Z 45



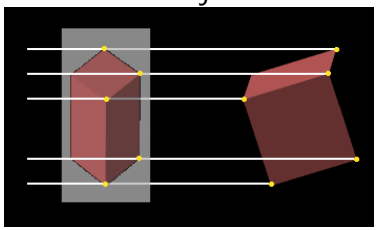
Hierarchy of Coordinate Systems

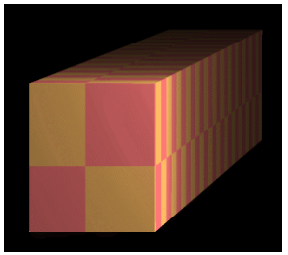
- Also called
  - Scene graph
  - Parent and child entities



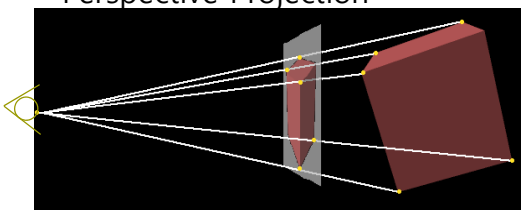
Camera – Parallel vs. Perspective Projection

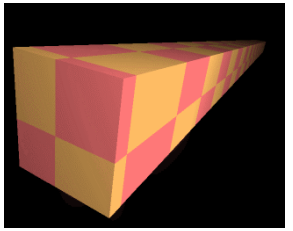
Parallel Projection



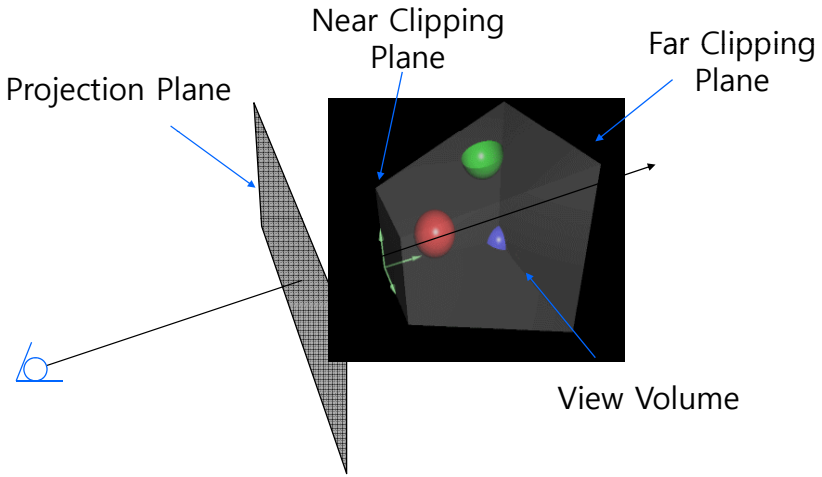


Perspective Projection





Camera – Perspective Projection

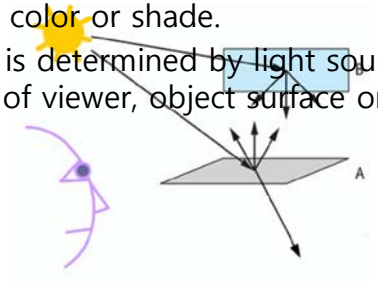


Camera Movement

- First-person camera
  - Game player controls the camera viewpoint.
  - Height is over the ground is determined by system.
  - Not use twist (except for, Descent game)
  - View volume is important for perspective projection.
    - It is affected by game player behind the monitor.
    - Zoom-in or zoom-out controls are unnatural and dangerous.
- Third-person camera
  - Tethered view from game player
  - Camera movement has to have a little bit of lags
  - View orientation changes can be done when the character is not moved or by the game player's controls
  - Be careful with camera's up-vector (i.e., twist)
  - Be careful with obstacles

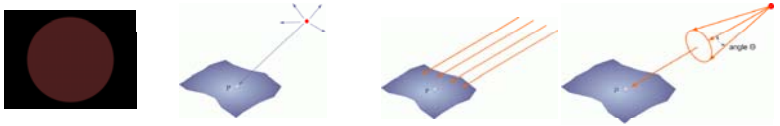
Lighting

- Starts from the lighting source
- Depending on the material properties on the object surface,
  - Absorption
  - Reflection
  - Transmission or Refraction
- Light-material interactions cause each point to have a different color or shade.
- Shading is determined by light source, object material, location of viewer, object surface orientation



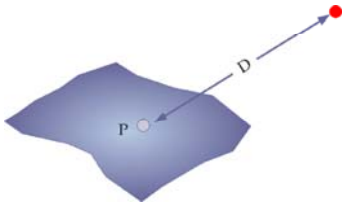
Light Source

- General light sources are difficult to work with because we must integrate light coming from all points on the source.
- Simple light sources
  - Ambient light – same amount of light everywhere in scene
  - Point light – model with position and color
  - Directional light – model with direction (parallel light)
  - Spot light – restrict light from ideal point source



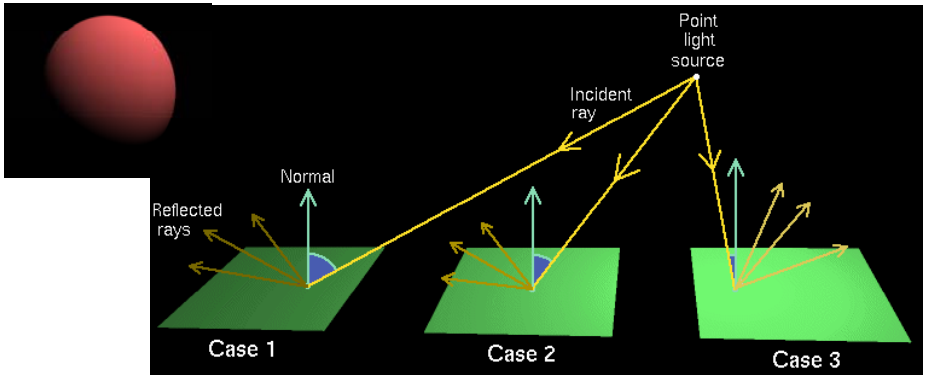
Ambient Reflection

- Ambient reflection gives basic, even illumination of all objects in a scene.



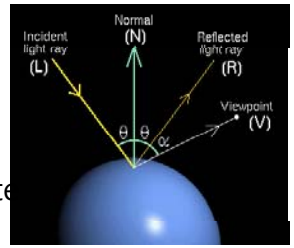
Diffuse Reflection

- The greater the angle between the normal and the vector from the point to the light source, the less light is reflected. Most light is reflected when the angle is 0 degrees, none is reflected at 90 degrees.

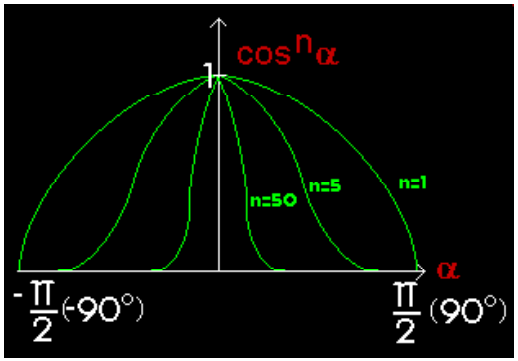


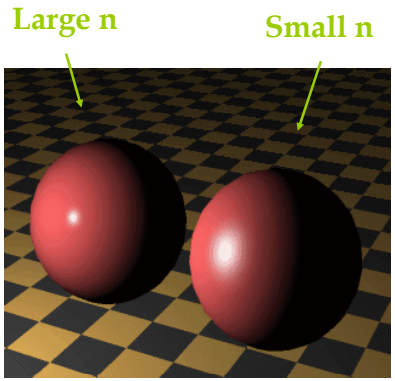
Specular Reflection (Phong Lighting Model)

- Maximum specular reflectance occurs when the viewpoint is along the path of the perfectly reflected ray (when *alpha* is zero).
- Specular reflectance falls off quickly as *alpha* increases.
- Falloff approximated by **cos^n(alpha)**.
- *n* varies from 1 to several hundred, depending on the material being modelled.
- n=1 provides broad, gentle falloff
- Higher values simulate sharp, focused highlight.
- For perfect reflector, n would be infinite



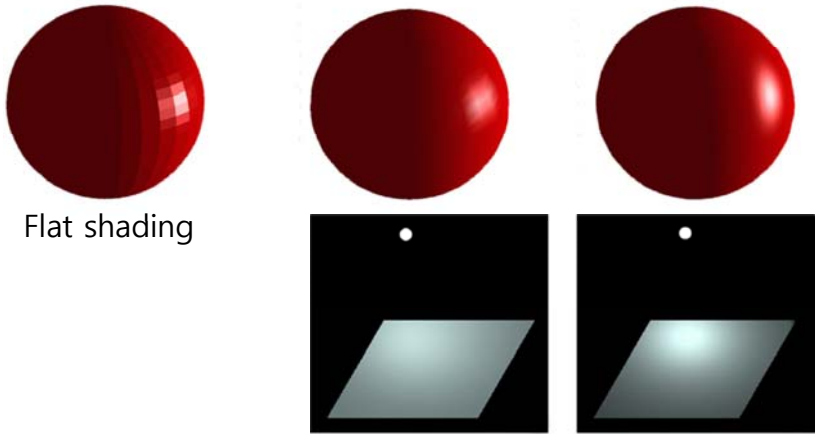
Fall off in Phong Shading





Flat, Gouraud, and Phong Shading

- Compare/contrast flat shading, Gouraud shading, and Phong shading



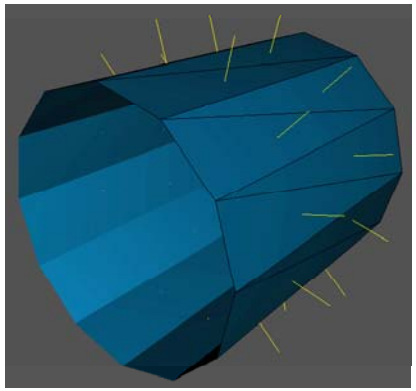
Flat shading

Gouraud shading

Phong shading

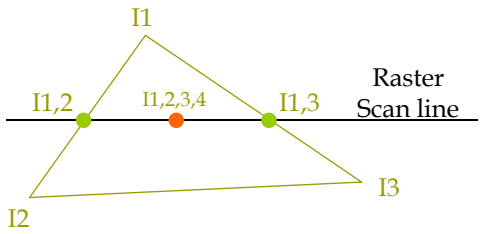
Flat Shading

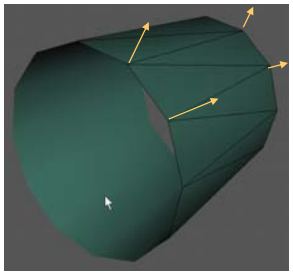
- Flat shading – each polygon face has a normal that is used to perform lighting calculations.



Gouraud Shading

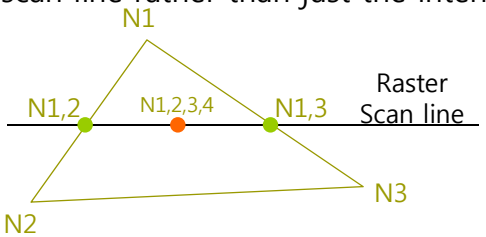
- Compute vertex normals by averaging face normals.
- Compute intensity at each vertex.
- Interpolate the intensity along the edges of the polygon.



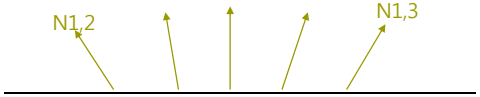


Phong Shading

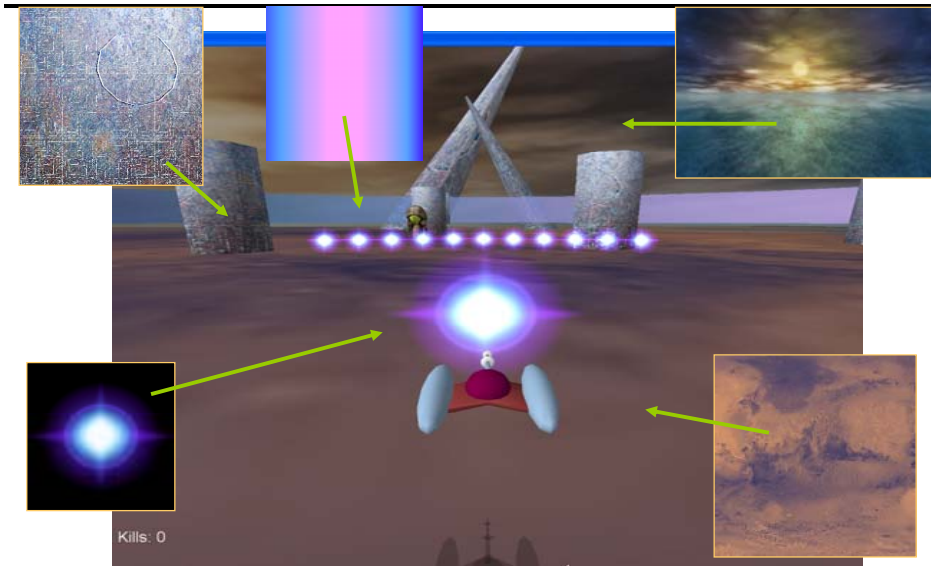
- Interpolate the Normals between the end points of the scan line rather than just the intensity



- Used with Phong lighting model to produce specular highlights.

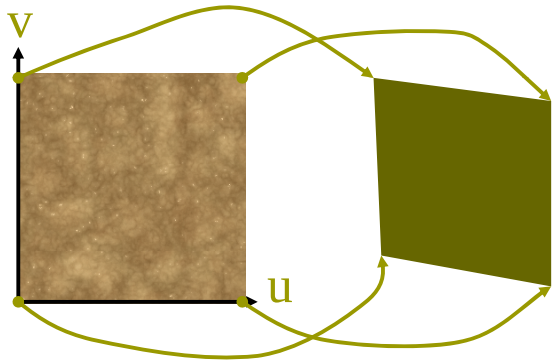


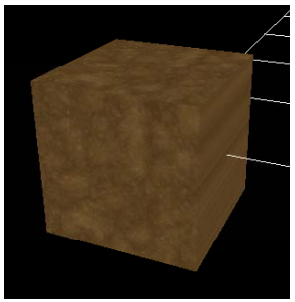
Texture Maps Used in Tank Game



Texture Mapping

- Apply picture to a polygon surface to add realism without explicitly creating geometry.

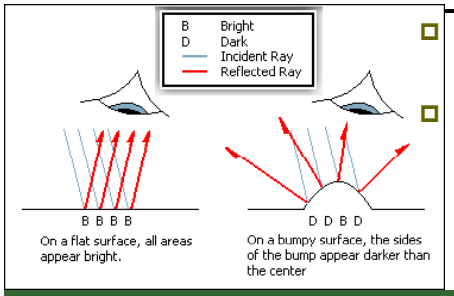




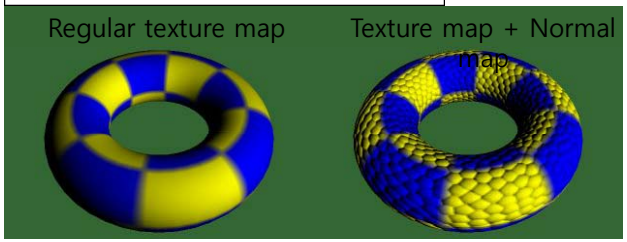
Texture mapping

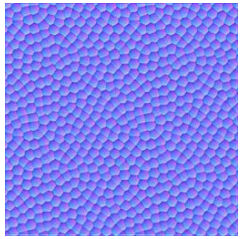
- Texture mapping techniques
  - **Color** (diffuse reflection coefficients) - **Texture mapping**
  - **Specular color** (to simulate specular light and reflection coefficients) - **Environment mapping** is a cheap way to create reflections (to render highly specular surface).
  - **Normal vector** (to simulate roughness and structure in materials) - **Bump mapping/Normal mapping** creates a realistic non-smooth surface by perturbing the normal vector (Do not perturb the surface geometry itself).
  - **Position** (normally only perpendicular to the surface) - **Displacement mapping**
  - **Transparency** (to get glass effects) (no refraction though)

Normal Mapping



- Modify normals on a per-pixel basis using a Normal Map
- The map is an R,G,B map where the 3 components represent the direction of the normal vector relative to the tangent of the polygon's surface



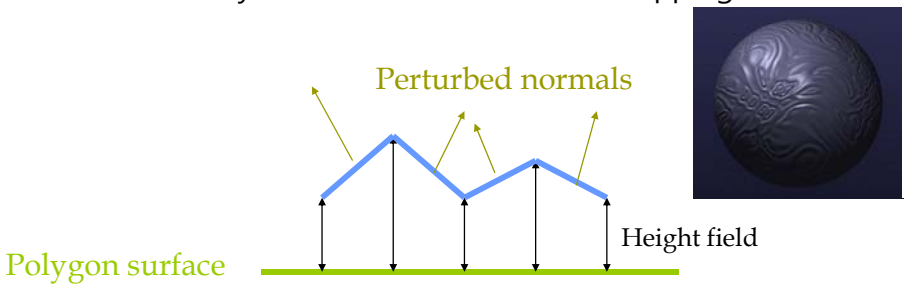


Normal Mapping

- In modern games, normal maps are computed for high polygon scenes and then mapped onto low polygon models giving them incredible levels of detail
- See 1st segment of UnReal Video on Normal Mapping

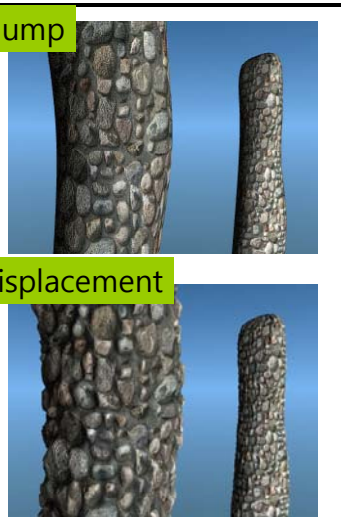
Bump Mapping

- Predecessor to Normal Mapping where a greyscale texture is used as a height field to modify the shading normal on the surface of a polygon (on a per pixel basis)
- Creates very similar effect as Normal Mapping



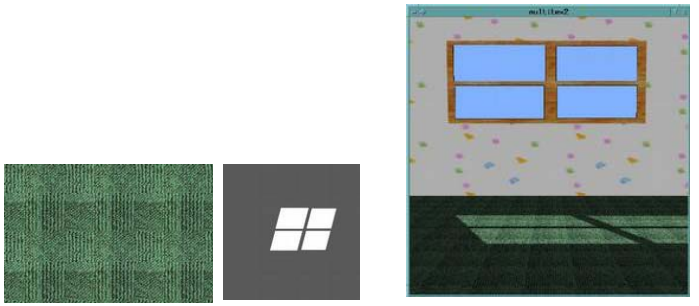
Displacement Mapping

- If you look at the polygon on its side, it will still look flat.
- Answer: Use Displacement Mapping to perturb the actual geometry of the mesh
- Height field is used to displace the geometry of the mesh along its normal
- See UnReal video on displacement map (right after Normal Map)



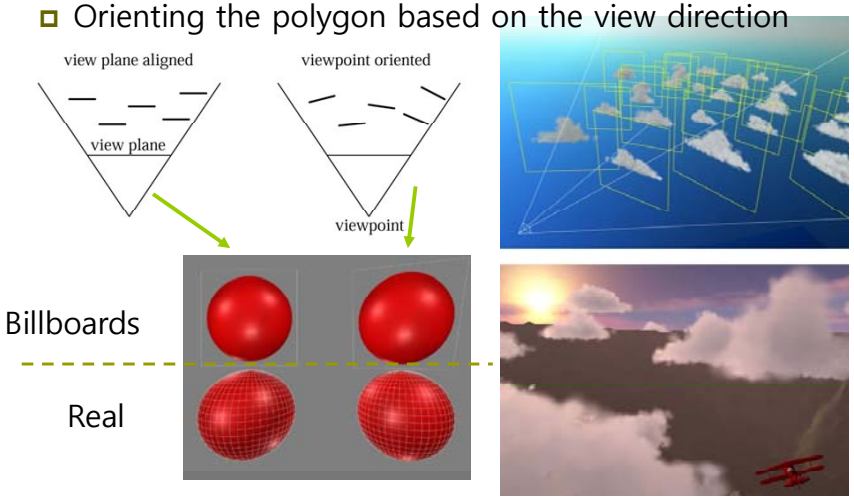
Multipass Rendering

- Blending allows results from multiple drawing passes to be combined together – e.g. lightmap effect



Billboarding

- Orienting the polygon based on the view direction



Modeling

- Levels
  - 2D hardware - pixels
  - 3D hardware - filled triangles with textures and Gouraud shading
  - Low-level 3D graphics library – same plus Phong shading, transformations, projections, clipping, etc
  - Graphics engines - levels of detail, curved primitives, etc
  - Modeling package tools - solid modeling, CSG, fractals, parametric surfaces, etc.

Polygonal Representations

- Polygon designs
  - Manual designs
  - 3D scanning
    - How to connect points of clouds
  - Interactive creation
    - Graphics primitives (sphere, cube, etc), extrusions, sweeping volumes
    - Mesh generation
    - Density dependent on curvature
    - Problems with crossing boundaries
  - High-level graphics primitives creation
    - Constructive solid geometry (CSG), Extrusions, Curves and surfaces

Polygonal representations

- Level of Detail (LOD)
  - Motivation
    - Mesh simplification
    - Level of detail approximation
    - Progressive transmission over the network
  - Problems
    - Which vertices should be thrown away
    - How to make geomorph
      - Use pixel-based geometry morphing
    - How to reduce the size of data for storage
    - How to improve the parts of the model selectively
  - Techniques
    - Resampling
    - Edge collapse/edge swaps
    - Vertex removal and retriangulation

Landscape/Terrains

- Terrains
  - Continuous
  - No object clipping
  - Difficult to use LOD
- Height fields (Height map)
  - Use a 2D bitmap (usually in grayscale image) to store height values, for rendering 3D mesh of terrains.
  - Height fields can be used in bump mapping to calculate where this 3D data would create shadow in a material; in displacement mapping to displace the actual geometric position of points over the texture surface.
- Fractal landscape generation
  - Supported by a modeling package
- Level of Detail (LOD) terrain algorithms
  - Use binary triangle tree-based ROAM(Real-time Optimally Adapting Meshes)
  - But, abrupt disappearing with peaks or valleys in LOD

Additional issues

- Shadows
  - Need shadow calculation for each light and object
  - Only used in the ground plane (the ground plane is treated as a separate object)
  - Shadow volumes
  - Shadow z-buffers
- Reflections
  - Reflection maps
  - Use stencil buffer for double rendering
- Shaders
  - HLSL, GLSL, Cg
  - Programmerable vertex or pixel

Efficiency

- Development of efficient game graphics is still very difficult.
- Hence, it requires thoughtful design.
  - Level of detail (LOD) techniques
  - Clipping
  - Portal techniques
  - Level design
  - Collision detection

Reference

- http://www.evl.uic.edu/spiff/class/cs426/
- http://dis.dankook.ac.kr/lectures/cg09/