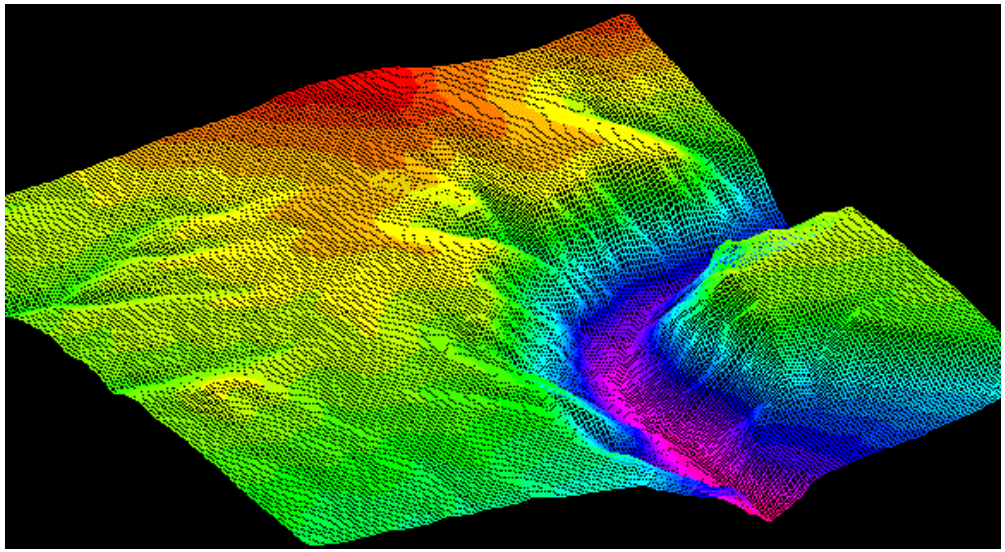
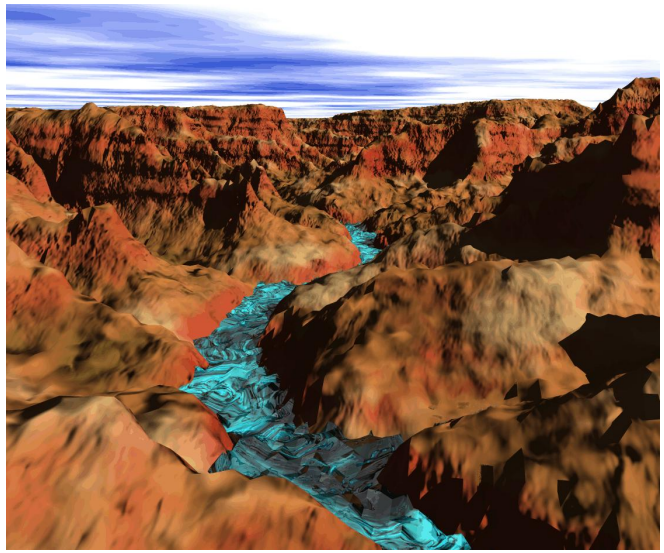
Design of Human Interface Game Software

- Modeling: Terrain
- Modeling: Scene Graph
- Buffers

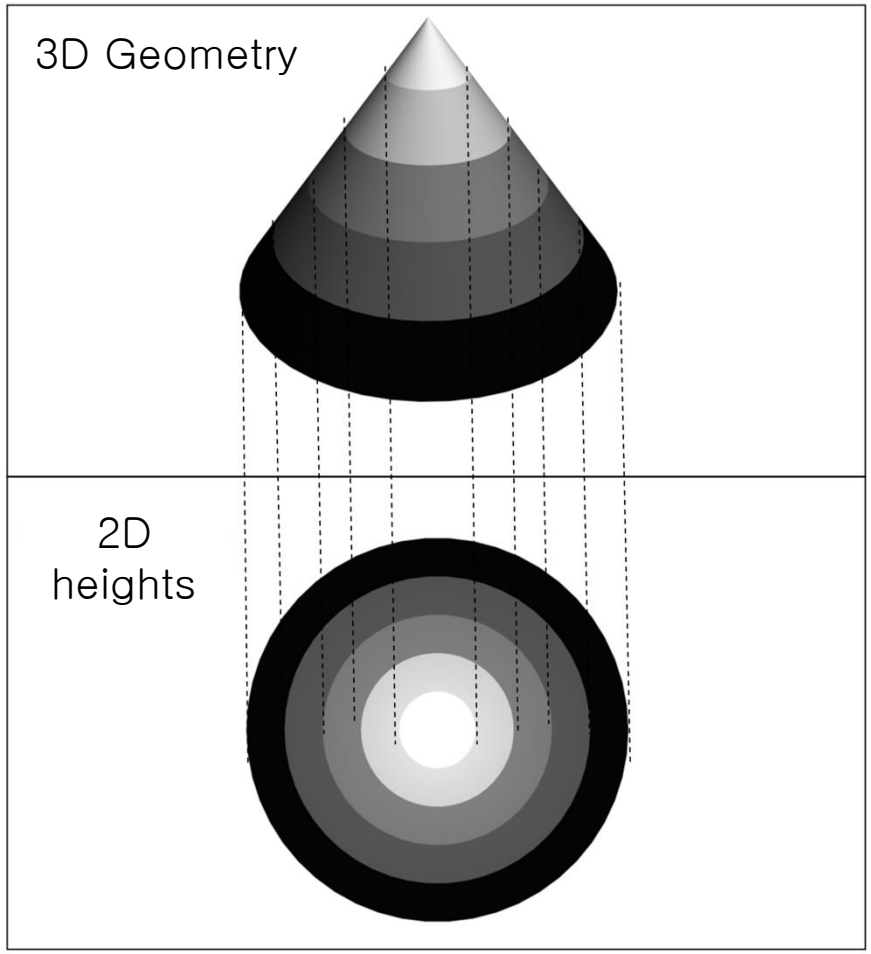
Terrain



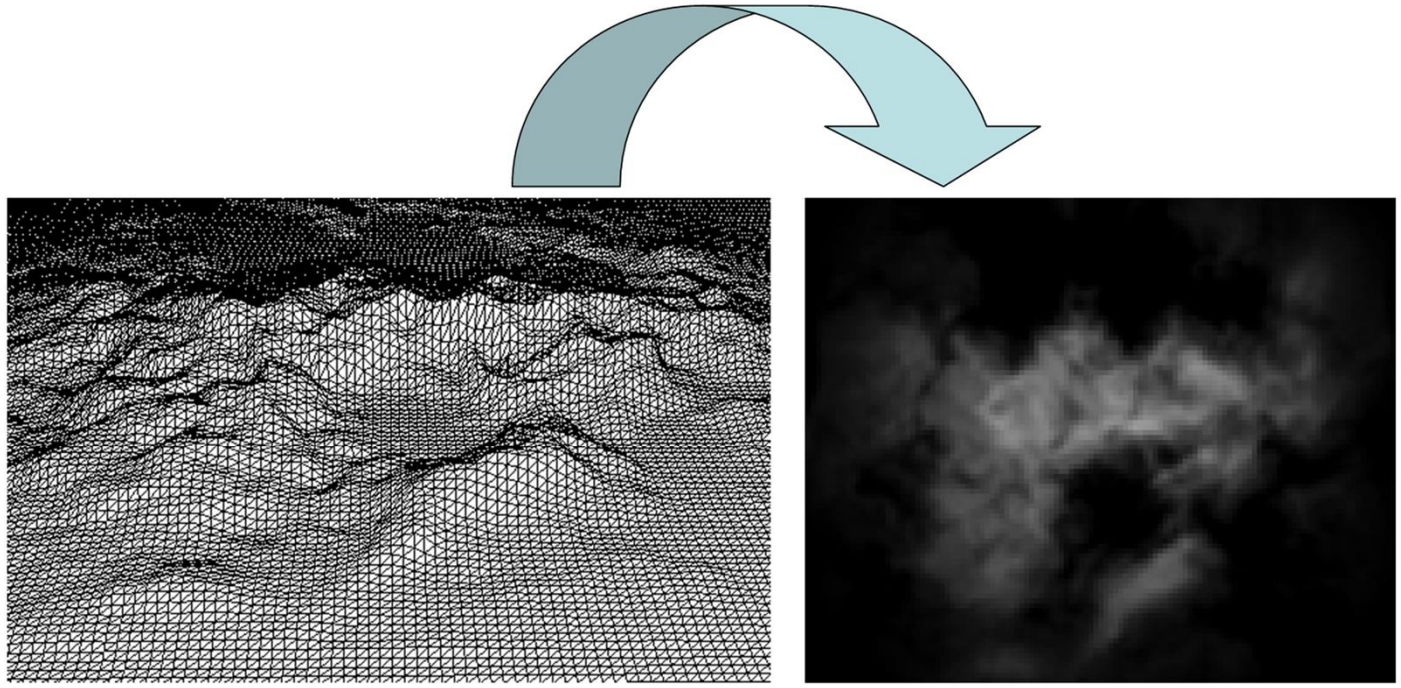


DEM (Digital Elevation Model)

- A height map is used to represent terrain or ground surface topography
- Height Map
  - A raster image used to store surface elevation data
  - Used in bump mapping, displacement mapping, terrains



DEM (Digital Elevation Model)



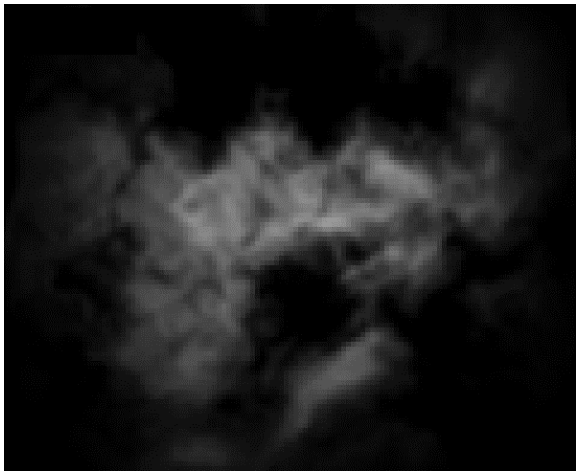
Original terrain

Height map

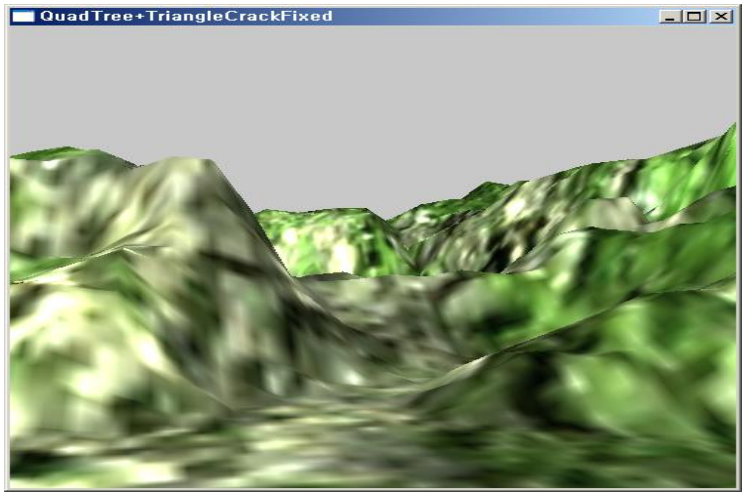
- Build a height map using remote sensing techniques
  - Radar scanning
  - Direct survey of land surface

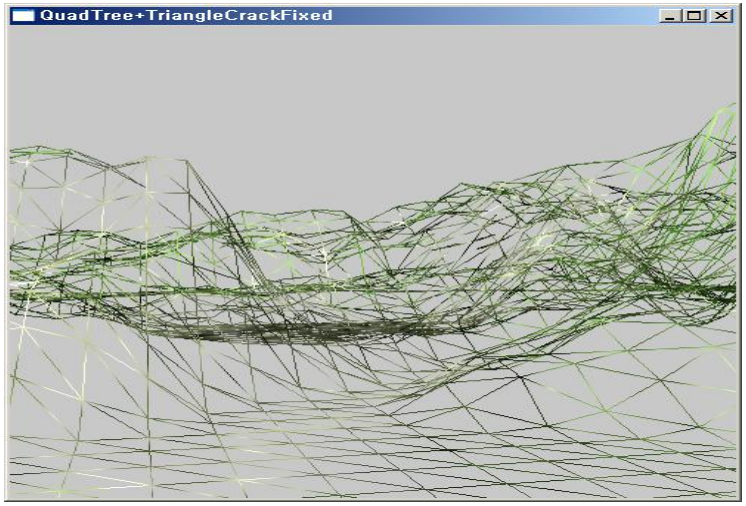
DEM (Digital Elevation Model)

- Construct a triangle mesh from a height map

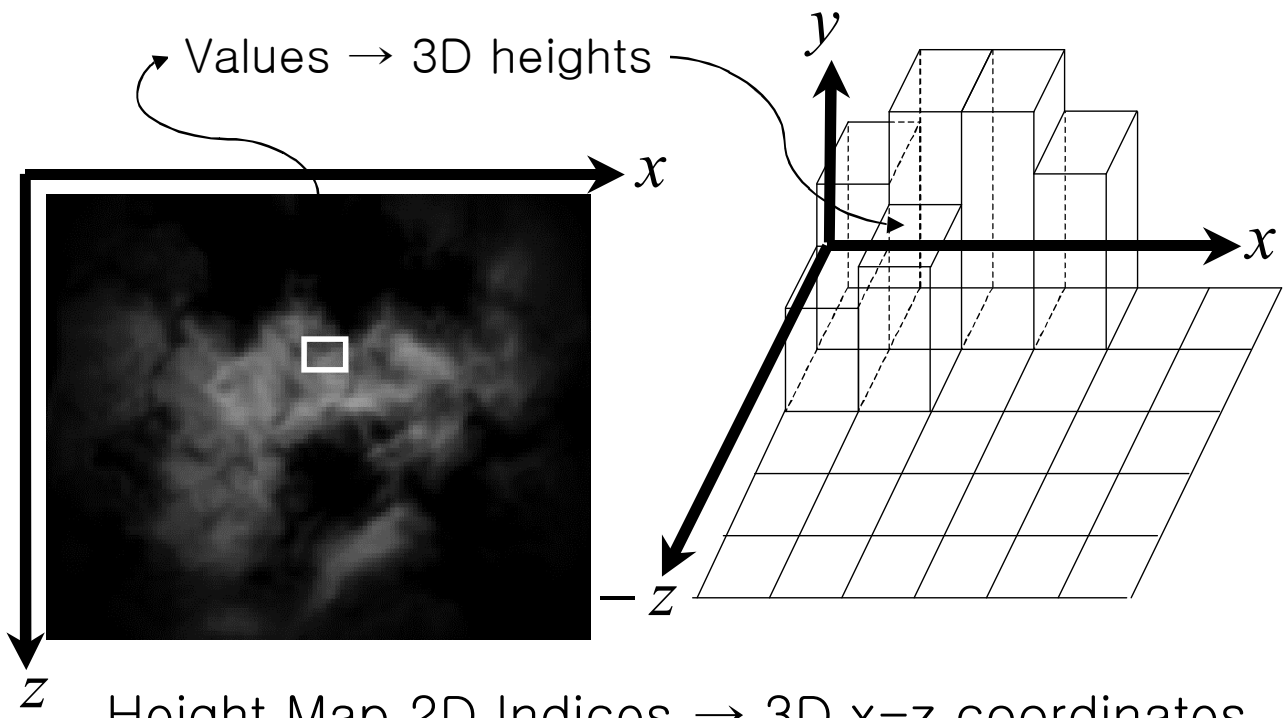


Height map





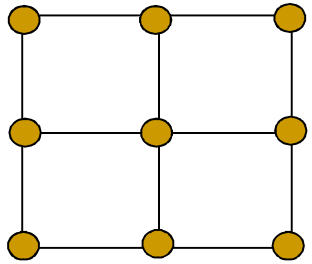
Constructing Triangle Meshes

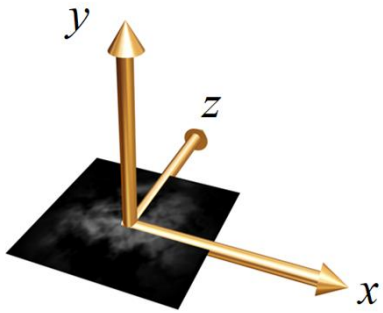


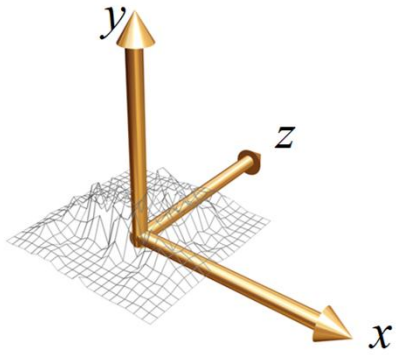
Height Map 2D Indices → 3D x-z coordinates
Height Map Values → 3D heights (y coordinate)
* Each cell generates two triangles

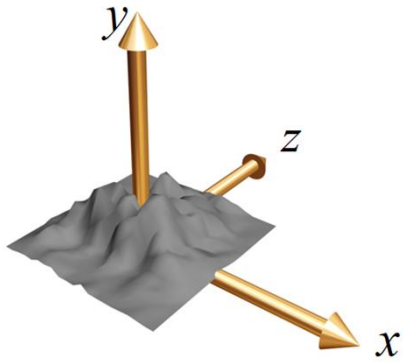
Constructing Triangle Meshes

- Vertices at the corners of the height map



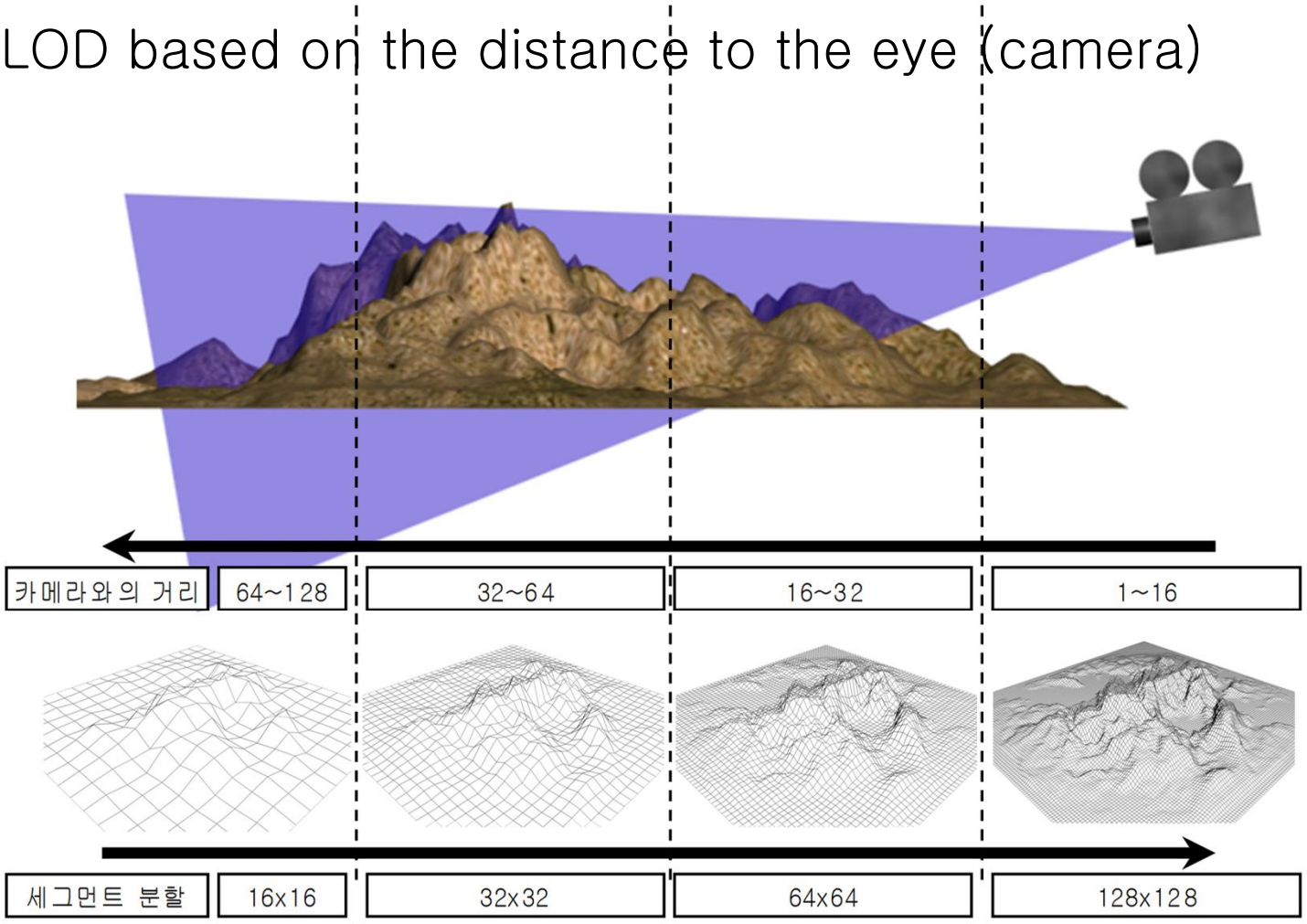






LOD(Level Of Detail)

- LOD based on the distance to the eye (camera)

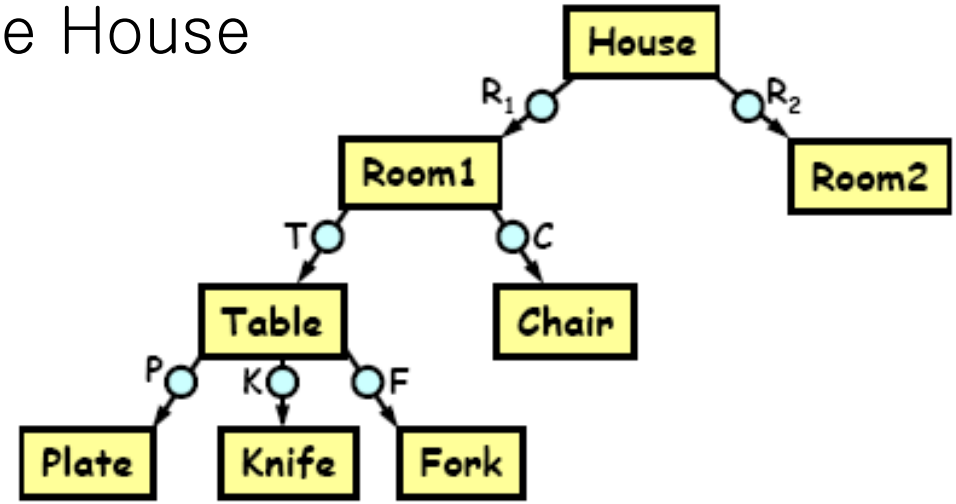


Scene Graph

- Scene Management
  - How to store virtual objects for efficient data access and rendering
- Hierarchical Structure
  - Good for culling – we can cull the whole sub-tree when the bounding box of a node is not visible
  - Apply transformations hierarchically

Scene Graph

- Local Coordinate System
  - Each object has its own local coordinate system
  - Each node stores the transformation to convert its local coordinates to parent's local coordinate system
- Ex) Transform vertex ***v*** of the Plate to the coordinates of the House
  - $(R_1TP)v$



Scene Graph

- Transformation of a vertex to the world coordinate system

$$v_{world} = M \times v_{local}$$

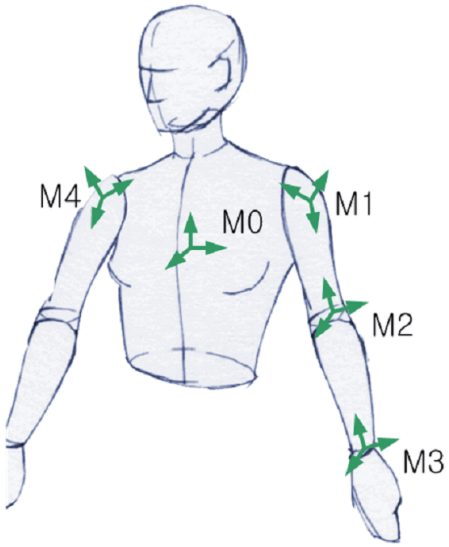
- In the hierarchy, $v_{world} = M_{parent} \times M_{child} \times v_{local}$
- For n parents.

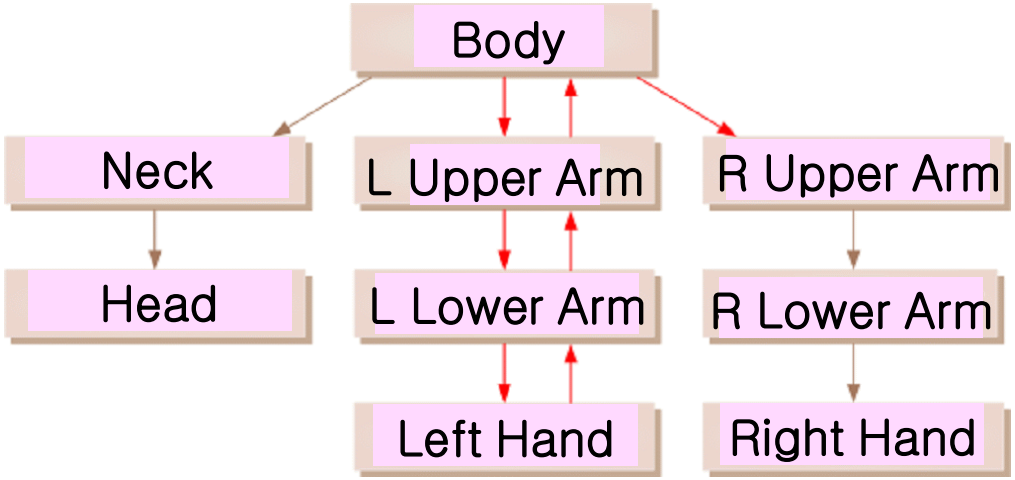
$$v_{world} = M_{parent\,(1)} \cdots \times M_{parent\,(n-1)} \times M_{parent\,(n)} \times M_{child} \times v_{local}$$

  - If the parent is transformed (moved), the children are also transformed (moved)

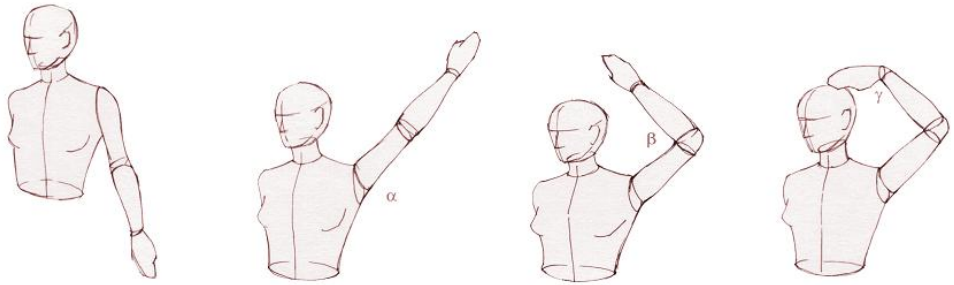
Scene Graph

- Traversing Hierarchy
  - **Push** for storing the current coordinate system
  - **Pop** for restoring the previous coordinate system





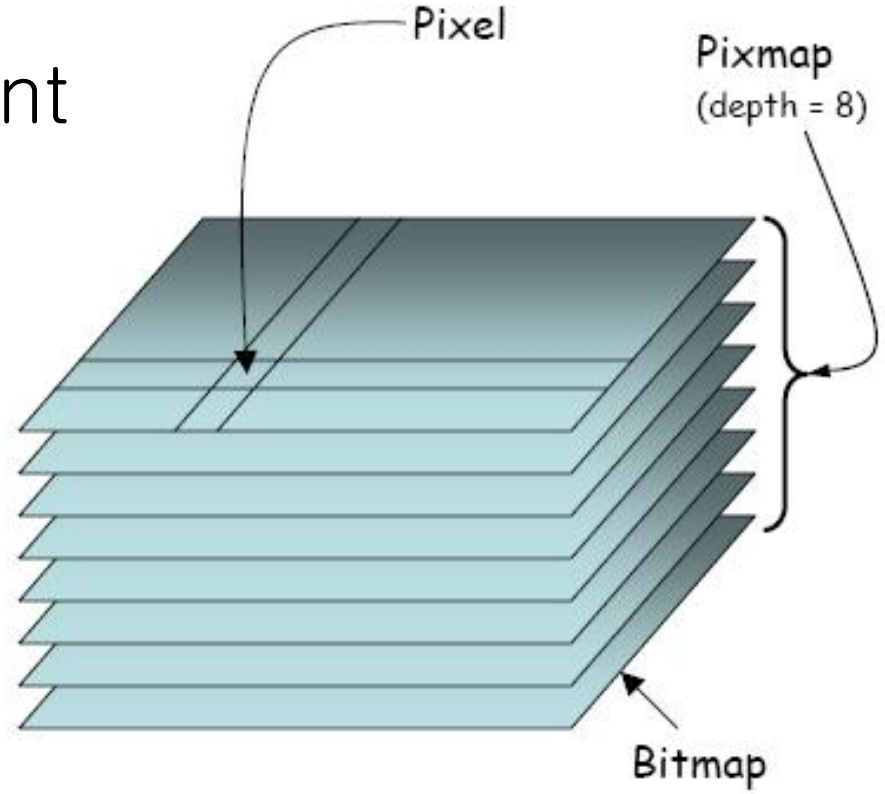
Scene Graph



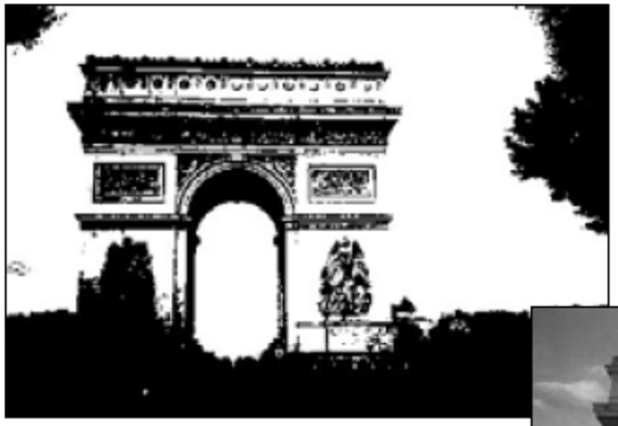
```
void drawArm( ){
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity( );                  World (Global) coordinate system
  Draw_Body( );                       body
  glPushMatrix( );                    Store world coordinate system
     GoToShoulderCoordinates( );      Shoulder local coordinate system
     Draw_UpperArm( );                upper arm
     glPushMatrix( );                 Store Shoulder coordinate system
       GoToElbowCoordinates( );        Elbow local coordinate system
       Draw_LowerArm( );              lower arm
       glPushMatrix( );               Store Elbow coordinate system
          GoToWristCoordinates( );    Wrist local coordinate system
          Draw_Hand( );               hand
       glPopMatrix( );                restore Elbow coordinate system
    glPopMatrix( );                   restore Shoulder coordinate system
 glPopMatrix( );                      restore World coordinate system
}
```

Bitmaps and Pixmaps

- Pixel: a picture element
- Bitmap
  - 2D array of single-bit pixels
- Pixmap
  - Stack of bitmaps
  - To represent full RGB color, it is sufficient to have 24-bit depth, 8 bits for each color channel



Bitmaps and Pixmaps



Bitmap (depth = 1)

Pixmap (depth = 8)

Pixmap (depth = 24)

Multi-Pass Rendering

- The pipeline takes one triangle at a time, so only local information and pre-computed maps are available
- Multi-pass techniques render the scene, or parts of the scene, multiple times
  - Makes use of auxiliary buffers to hold information
  - Make use of tests and logical operations on values in the buffers
  - Really, a set of functionality that can be used to achieve a wide range of effects
    - Mirrors, shadows, bump-maps, anti-aliasing, compositing, …

Buffers

- Buffers allow you to store **global information** about the rendered scene
  - Like scratch work space, or extra screen memory
  - They are only cleared when you say so
  - This functionality is fundamentally different from that of vertex or pixel shaders
- Buffers are defined by:
  - The type of **values** they store
  - The logical **operations** that they influence
  - The way they are **accessed** (written and read)

OpenGL Buffers

- Color buffers: Store RGBA color information for each pixel
  - OpenGL actually defines four or more color buffers: front/back (double buffering), left/right (stereo) and auxiliary color buffers
- Depth buffer: Stores distance to object for each pixel (Z-buffer)
- Stencil buffer: Masking operations
- Accumulation buffer: Composition of images

Stencil Buffer

- The stencil buffer acts like a paint stencil – it lets some fragments through but not others
- It stores multi-bit values – you have some control of #bits
- You specify two things:
  - The test that controls which fragments get through
  - The operations to perform on the buffer when the test passes or fails
  - **All tests/operation look at the value in the stencil that corresponds to the pixel location of the fragment**
- Typical usage: One rendering pass sets values in the stencil, which control how various parts of the screen are drawn in the second pass

Stencil Tests

- You give an operation, a reference value, and a mask
- Operations:
  - Always let the fragment through
  - Never let the fragment through
  - Logical operations between the reference value and the value in the buffer: <, <=, =, !=, >, >=
- The mask is used to select particular bit-planes for the operation
  - `(reference & mask ) op ( buffer & mask )`

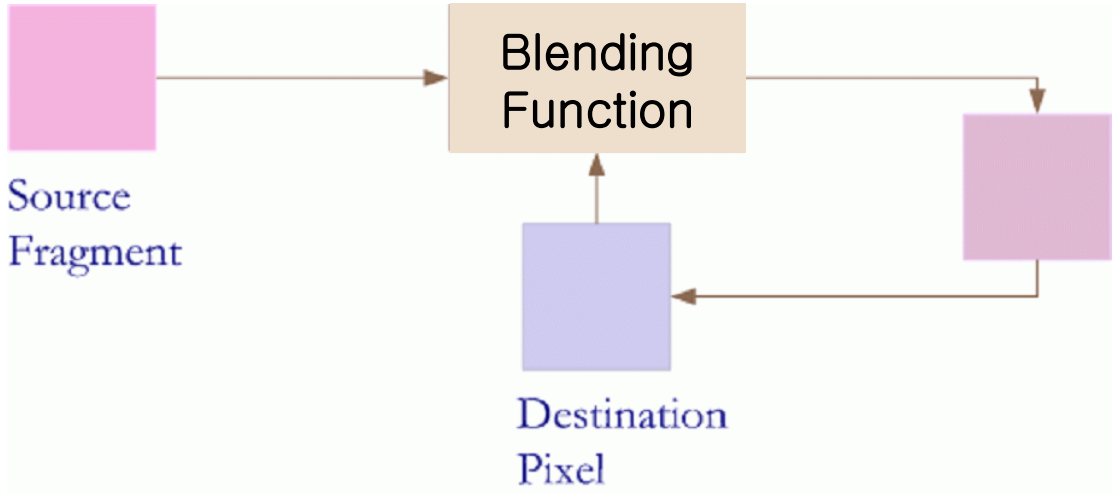
Chroma Keying

+

=

Blending

- Blend fragments and the pixels in the frame buffer
  - Input fragment = Source Fragment
  - Pixel in the frame buffer = Destination Pixel
  - Result is stored in the destination pixel.



Blending

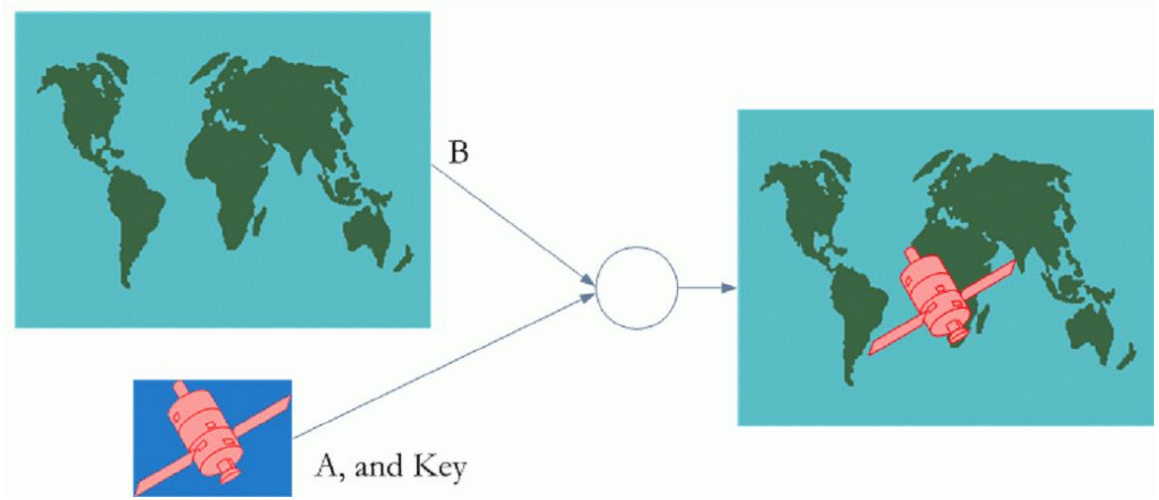
- (R_s, G_s, B_s, A_s) : Source fragment color
- (R_d, G_d, B_d, A_d) : Destination pixel color
- (S_r, S_g, S_b, S_a) : Blending factor for source fragment
- (D_r, D_g, D_b, D_a) : Blending factor for destination pixel

$$(R, G, B, A) = (R_s S_r + R_d D_r,\ G_s S_g + G_d D_g,\ B_s S_b + B_d D_b,\ A_s S_a + A_d D_a)$$

- Various blending factors
  - (GL_ONE, GL_ZERO)
  - (GL_SRC_ALPHA, GL_ONE)
  - (GL_ONE, GL_DST_ALPHA)
  - (GL_SRC_ALPHA, GL_DST_ALPHA)

Blending

- Image Composition
  - Blend the image with the background. Make the blue area of the image A transparent
- Image A = source fragment, background B = destination
- Make the alpha value of blue fragment to be 0. Set the source factor as GL_SRC_ALPHA. Then the factor for the blue fragment is (0, 0, 0, 0)



Blending

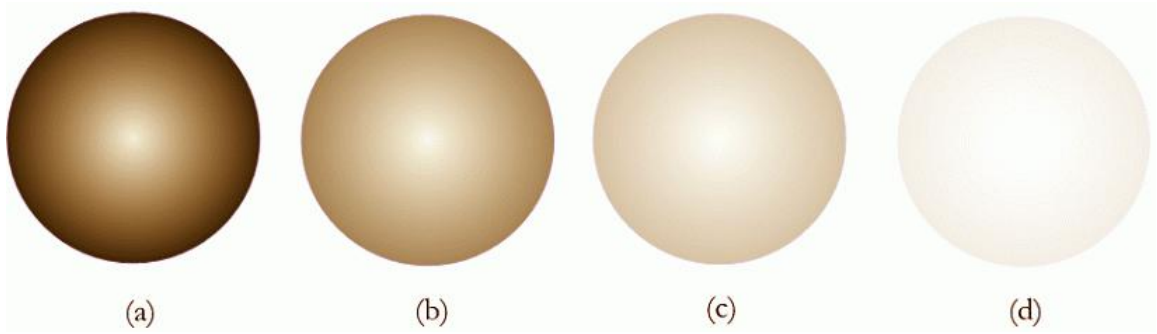
- Choosing transparent area = Keying, Blue Screening
  - Use specific color: Chroma Keying
  - Use brightness: Luma Keying
  - Select specific shape: Matte, Matting

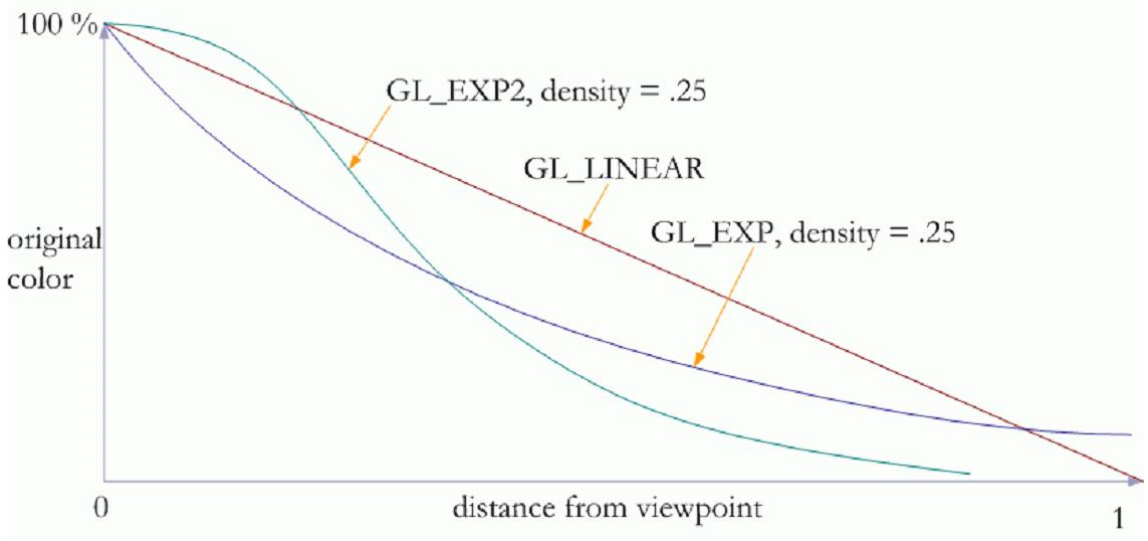
Depth Cue

- Perspective projection: objects in a distance are small
- Repeated patterns: more patterns in a distance
- Occlusion
- Shading and shadows

Fog Effect

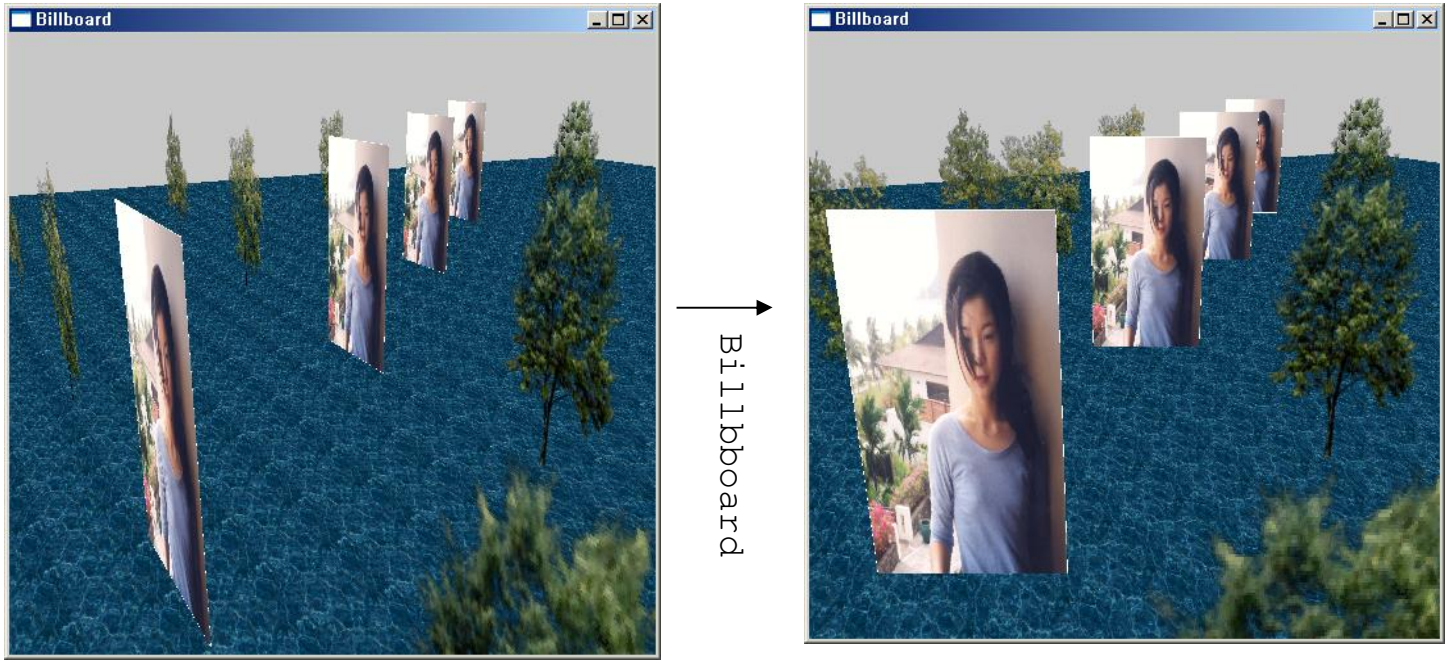
- Objects in a distance are diluted.





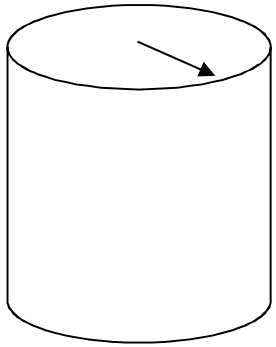
Billboard

- A billboard is used to represent a complex object with a texture mapping. Billboards are always facing the eye (camera). The texture on the billboard is blended with the background.



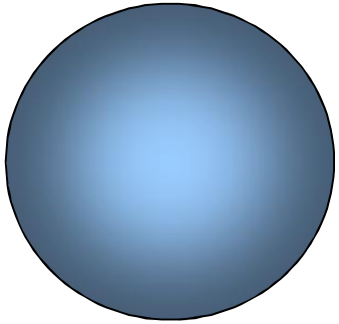
Billboard

- To make the billboard facing the eye, we need to compute the rotation matrix. The rotation matrix is the inverse matrix of the rotation part of the eye transformation.



• Cylinderical Billboards

Consider the rotations around Y axis



• Spherical Billboards

Consider every rotations