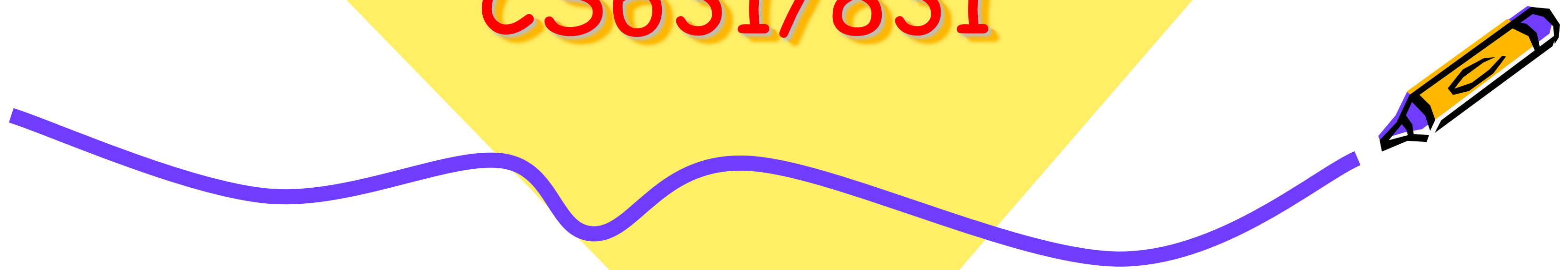
Computer Graphics Foundation to Understand Game Engine CS631/831

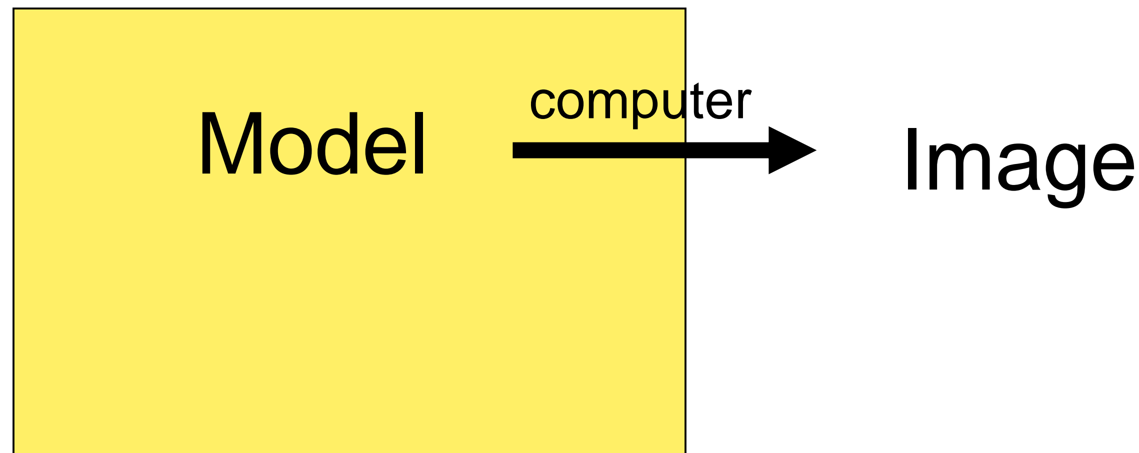


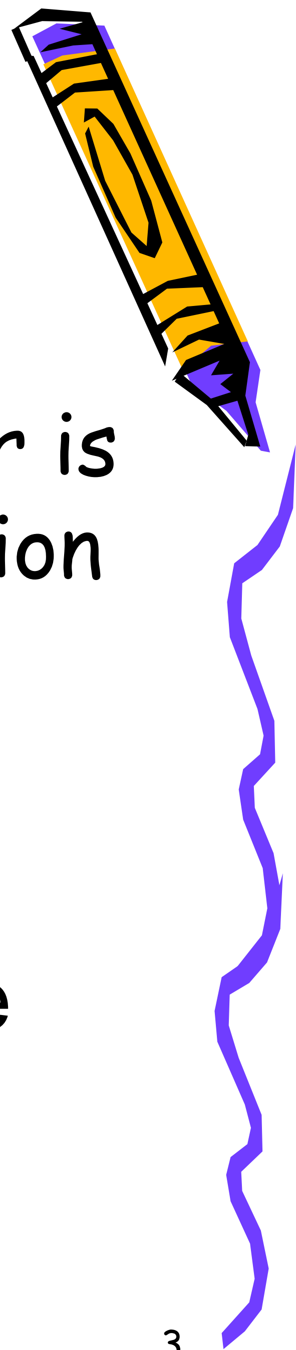
Quick Recap

- Computer Graphics is using a computer to generate an image from a representation.

Modeling

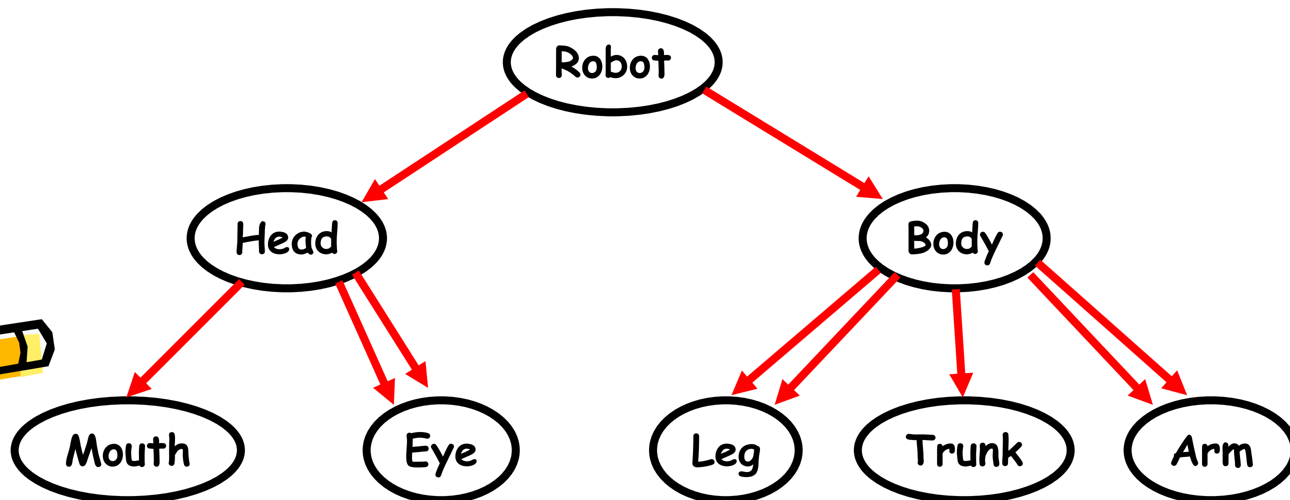
- What we have been studying so far is the mathematics behind the creation and manipulation of the 3D representation of the object.

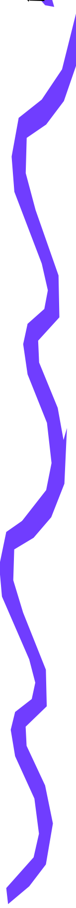




Modeling: The Scene Graph

- The *scene graph* captures transformations and object-object relationships in a DAG
- Objects in black; blue arrows indicate instancing and each have a matrix





Modeling: The Scene Graph

- *Traverse* the scene graph in depth-first order, concatenating transformations
- Maintain a *matrix stack* of transformations

Visited

Unvisited

Active

Matrix Stack

Robot

Head

Body

Mouth

Eye

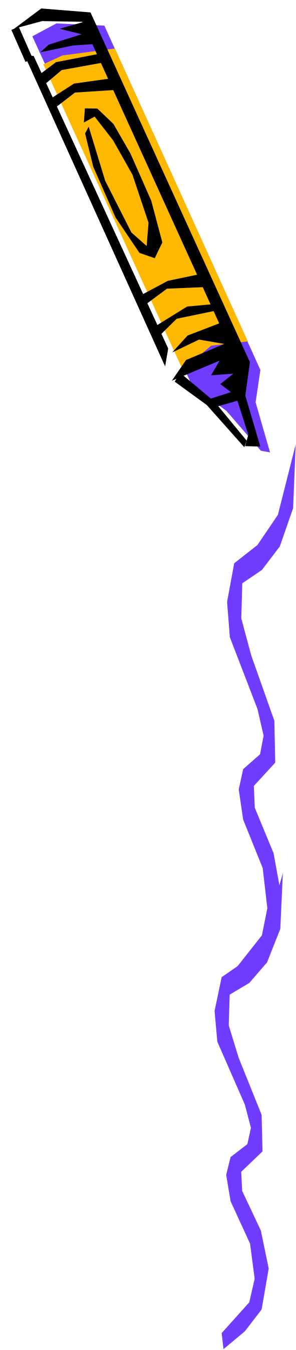
Leg

Trunk

Arm

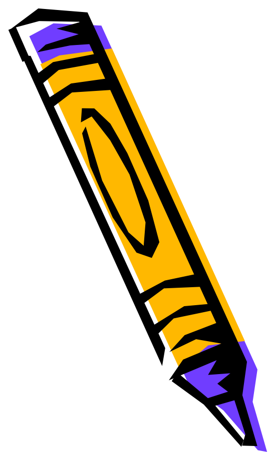
Foot

Motivation for Scene Graph

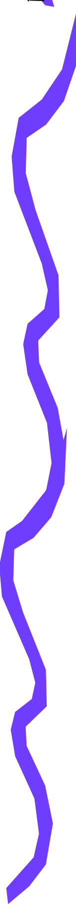


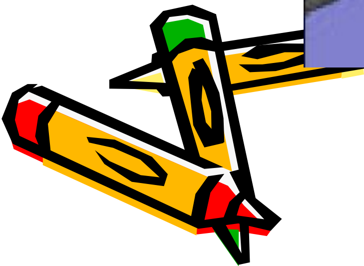
- Three-fold
  - Performance
  - Generality
  - Ease of use
- How to model a scene ?
  - Java3D, Open Inventor, Open Performer, VRML, etc.

Scene Graph Example

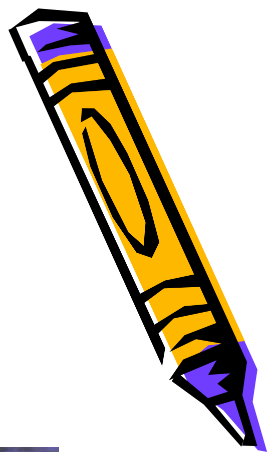


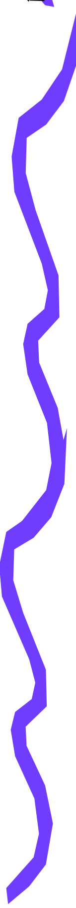
Scene Graph Example

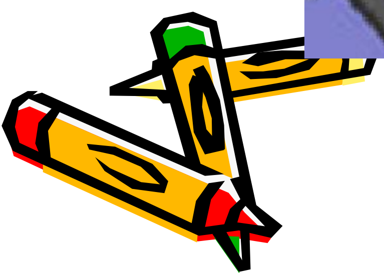




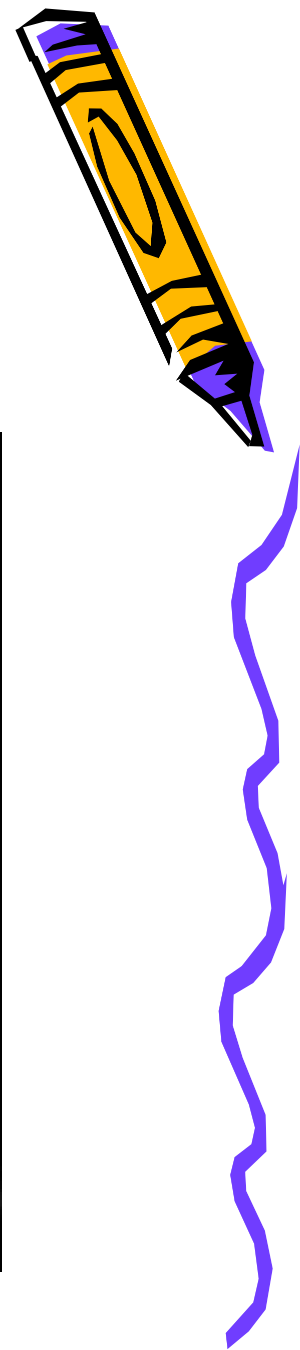
Scene Graph Example

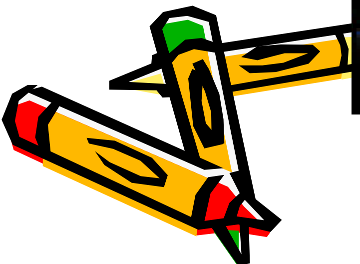


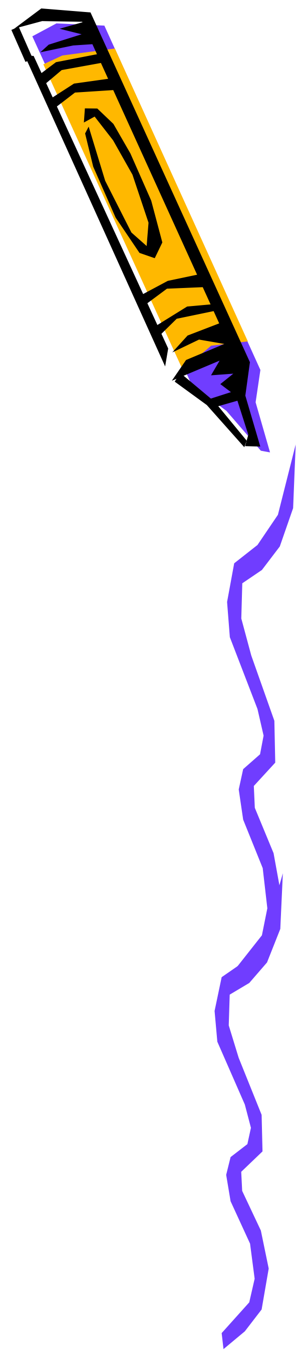




Scene Graph Example



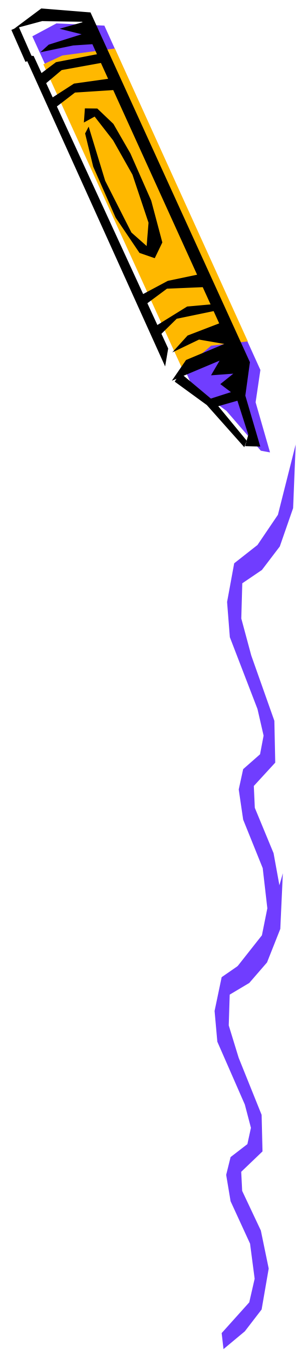




Scene Description

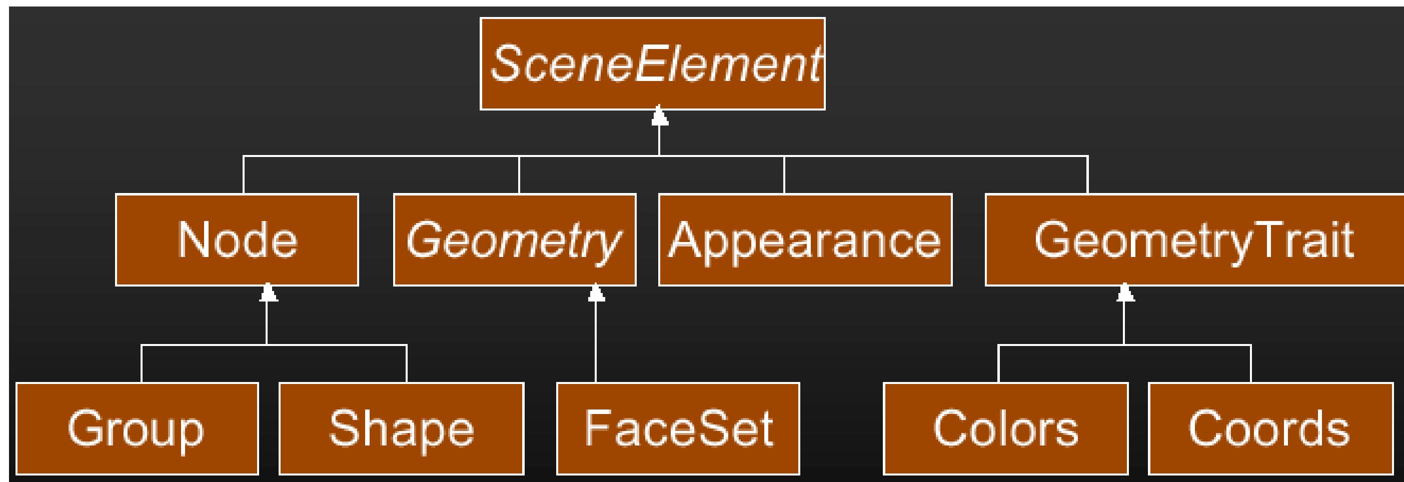
- Set of Primitives
- Specify for each primitive
  - Transformation
  - Lighting attributes
  - Surface attributes
    - Material (BRDF)
    - Texture
    - Texture transformation

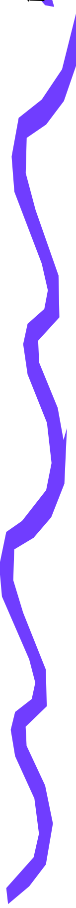
Scene Graphs



- Scene Elements
  - Interior Nodes
    - Have children that inherit state
    - transform, lights, fog, color, …
  - Leaf nodes
    - Terminal
    - geometry, text
  - Attributes
    - Additional sharable state (textures)

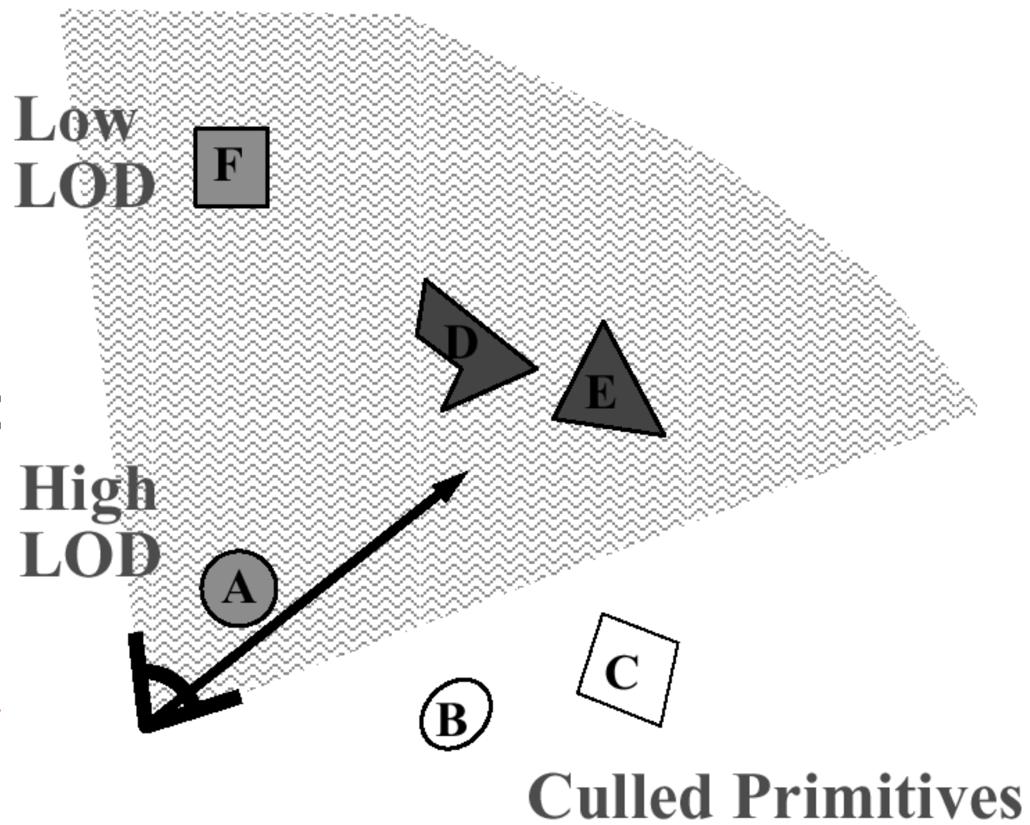
Scene Element Class Hierarchy

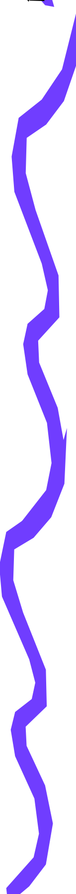




Scene Graph Traversal

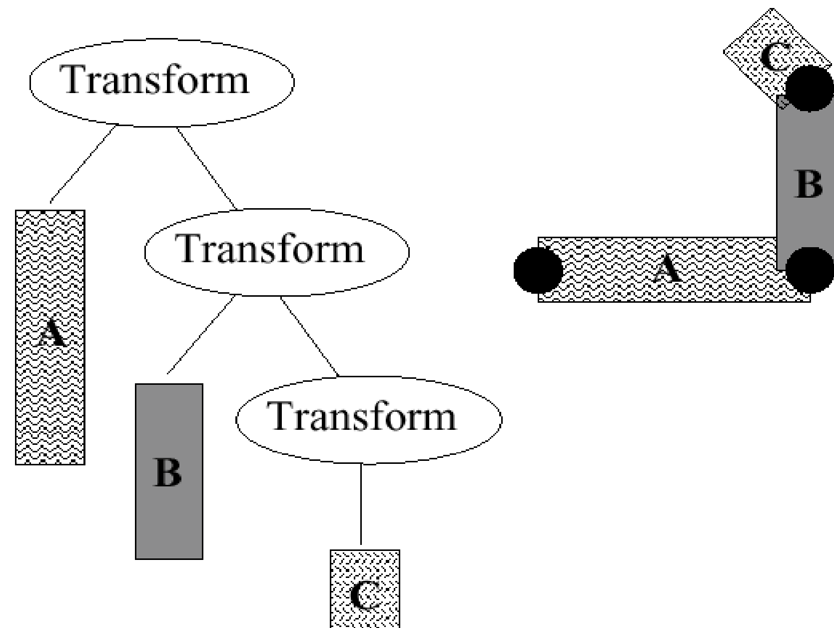
- Simulation
  - Animation
- Intersection
  - Collision detecti
  - Picking
- Image Generatic
  - Culling
  - Detail elision
  - Attributes

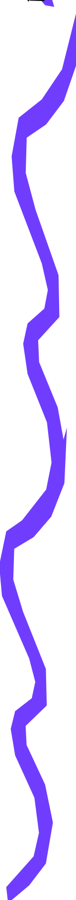




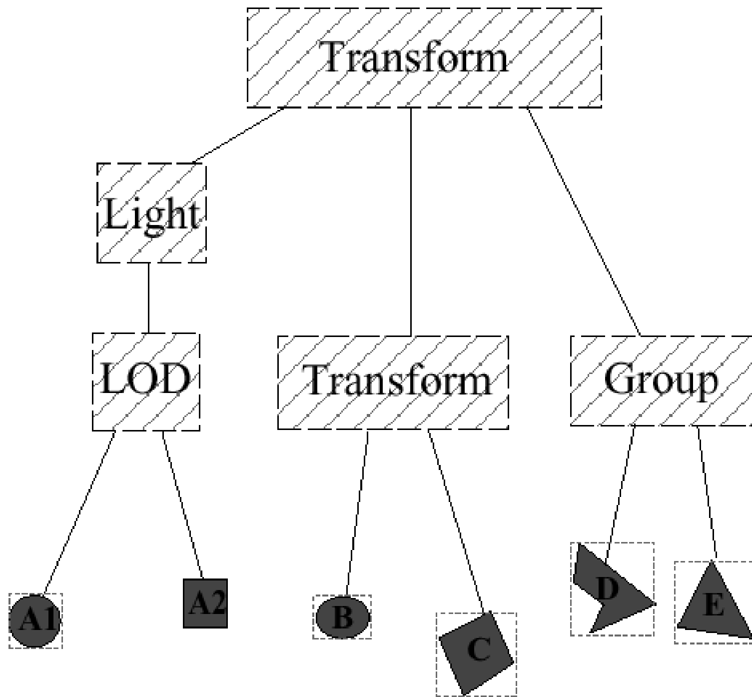
Functional Organization

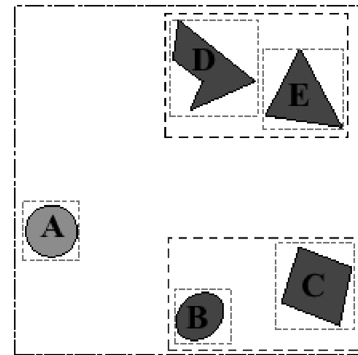
- Articulated Transformations
  - Animation
  - Difficult to optimize animated objects

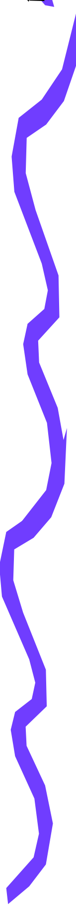




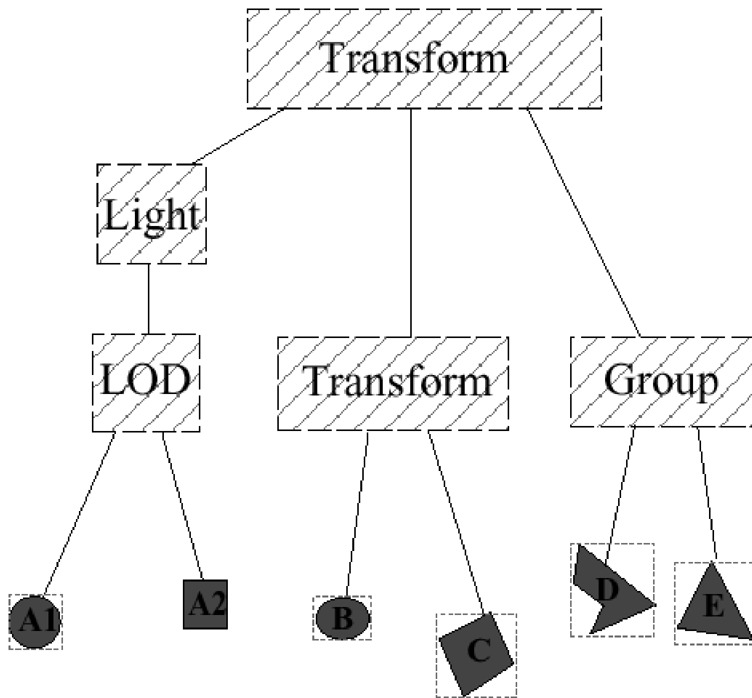
Bounding Volume Hierarchies

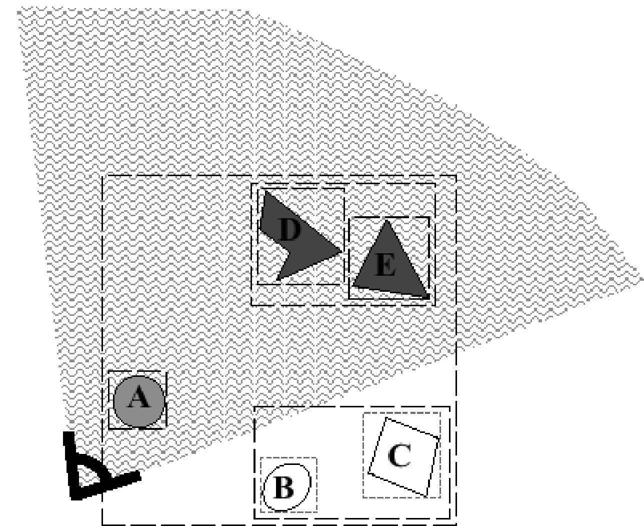


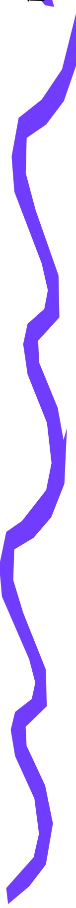




View Frustum Culling

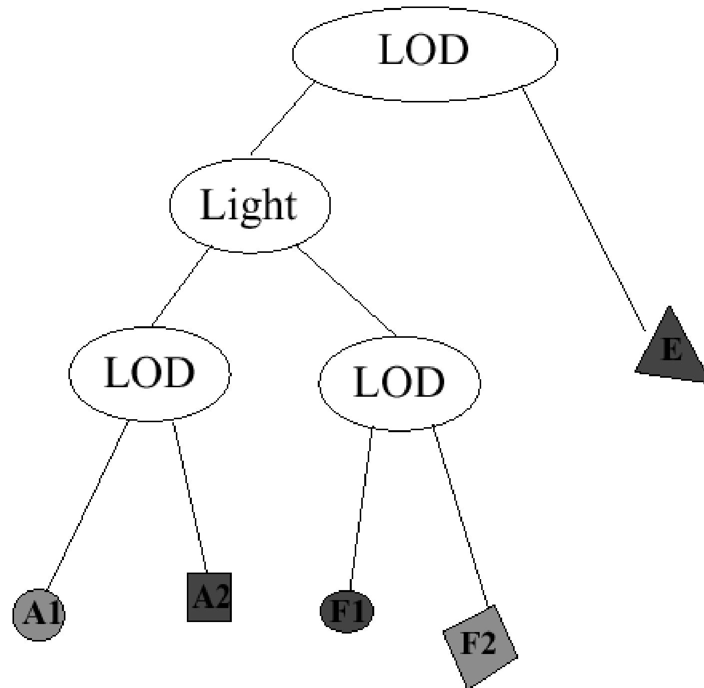


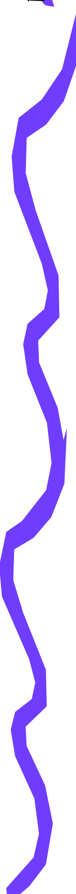




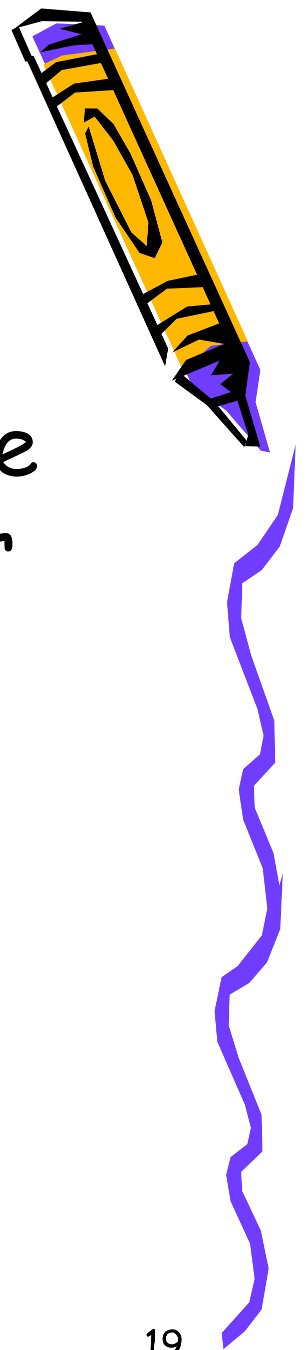
Level Of Detail (LOD)

- Each LOD nodes have distance ranges





What is a Transformation?

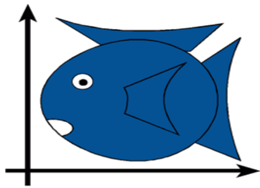


- Maps points (x, y) in one coordinate system to points (x', y') in another coordinate system

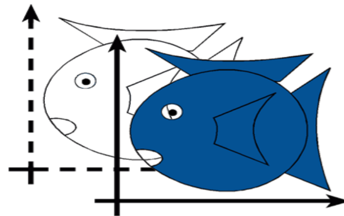
$$x' = ax + by + c$$
$$y' = dx + ey + f$$

Transformations

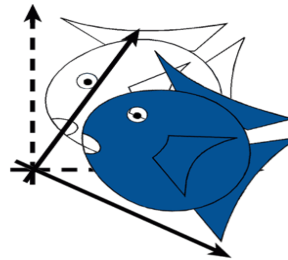
- Simple transformation
  - Translation
  - Rotation
  - Scaling



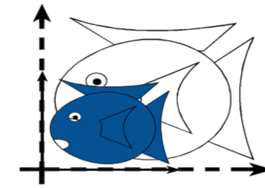
Identity



Translation

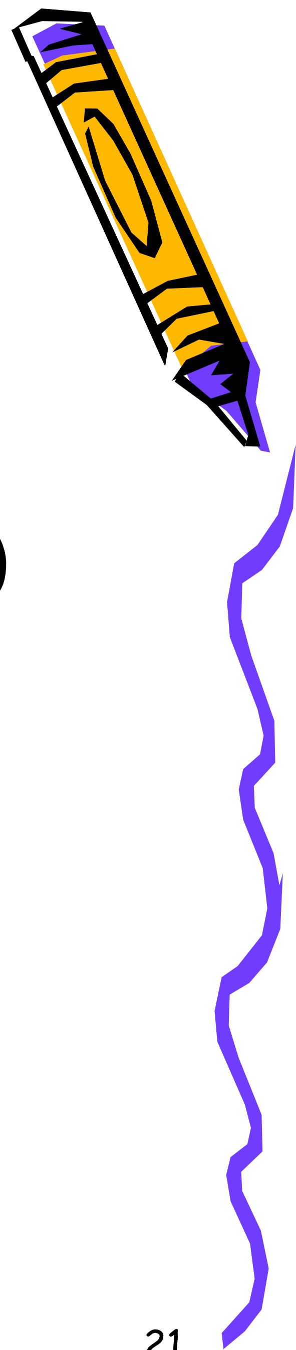


Rotation



Isotropic (Uniform) Scaling

Transformations



- Why use transformations?
  - Position objects in a scene (modeling)
  - Change the shape of objects
  - Create multiple copies of objects
  - Projection for virtual cameras
  - Animations

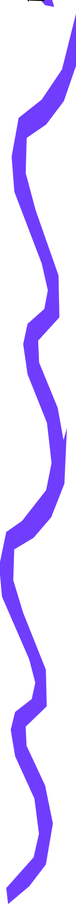
How are Transforms Represented?

$$x' = ax + by + c$$

$$y' = dx + ey + f$$

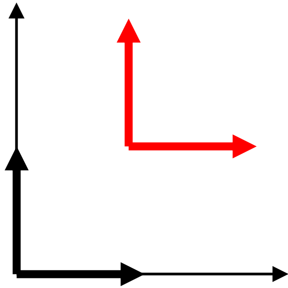
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ d & e \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} c \\ f \end{bmatrix}$$

$$p' = M p + t$$

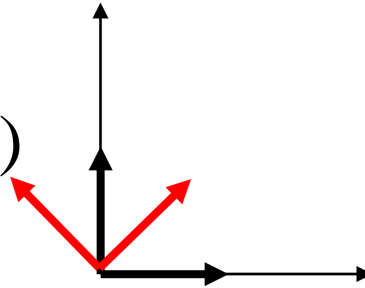


Combining Translation & Rotation

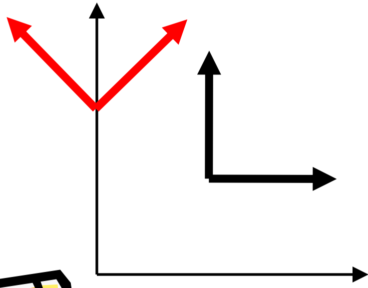
$T(1,1)$



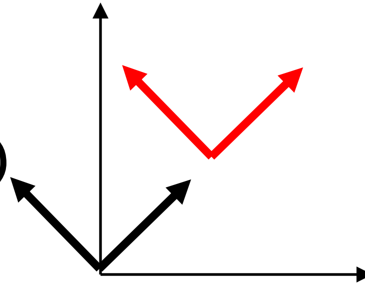
$R(45°)$



$R(45°)$



$T(1,1)$



Combining Translation & Rotation

$$\mathbf{v}' = \mathbf{v} + T$$

$$\mathbf{v}'' = R\mathbf{v}'$$

$$\mathbf{v}'' = R(\mathbf{v} + T)$$

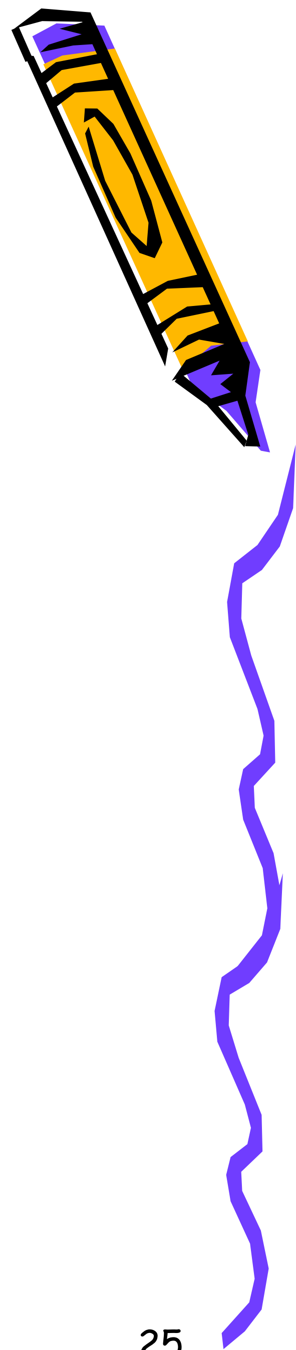
$$\mathbf{v}'' = R\mathbf{v} + RT$$

$$\mathbf{v}' = R\mathbf{v}$$

$$\mathbf{v}'' = \mathbf{v}' + T$$

$$\mathbf{v}'' = R\mathbf{v} + T$$

Homogeneous Coordinates

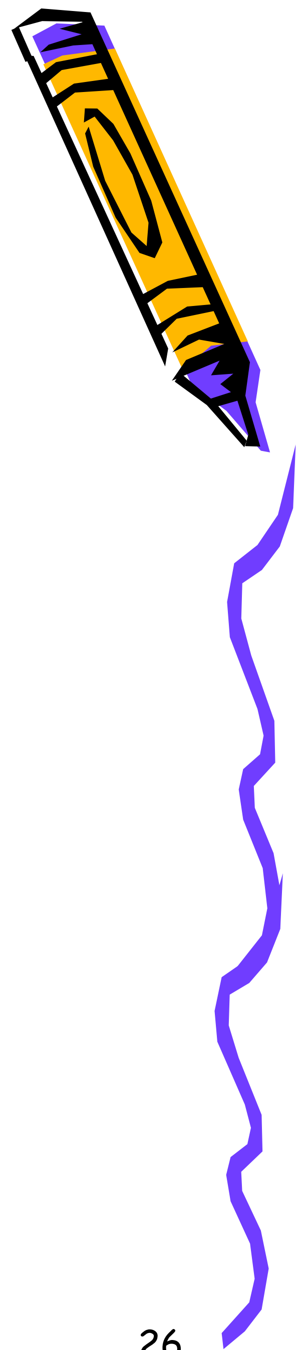


- Add an extra dimension
  - in 2D, we use 3 x 3 matrices
  - In 3D, we use 4 x 4 matrices
- Each point has an extra value, *w*

$$\begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix} = \begin{pmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix}$$

$$p' = \mathbf{M}\, p$$

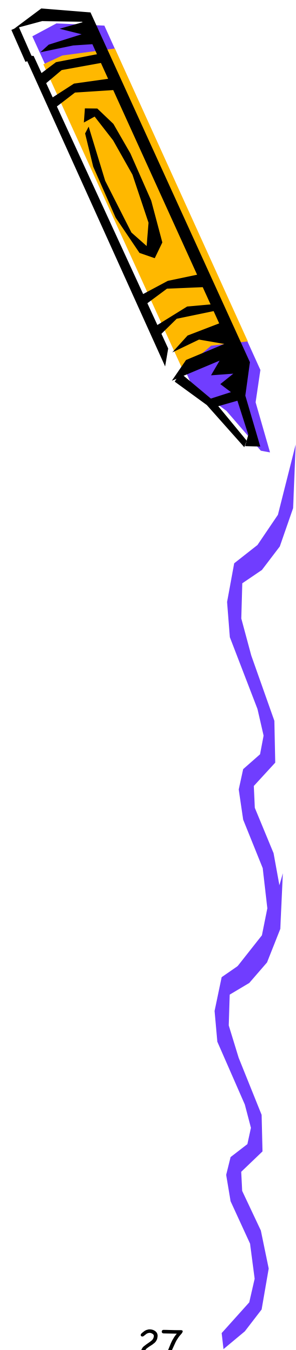
Homogeneous Coordinates



- Most of the time w = 1, and we can ignore it

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Combining Transformations



$$\mathbf{v}' = S\mathbf{v}$$

$$\mathbf{v}'' = R\mathbf{v}' = RS\mathbf{v}$$

$$\mathbf{v}''' = T\mathbf{v}'' = TR\mathbf{v}' = TRS\mathbf{v}$$

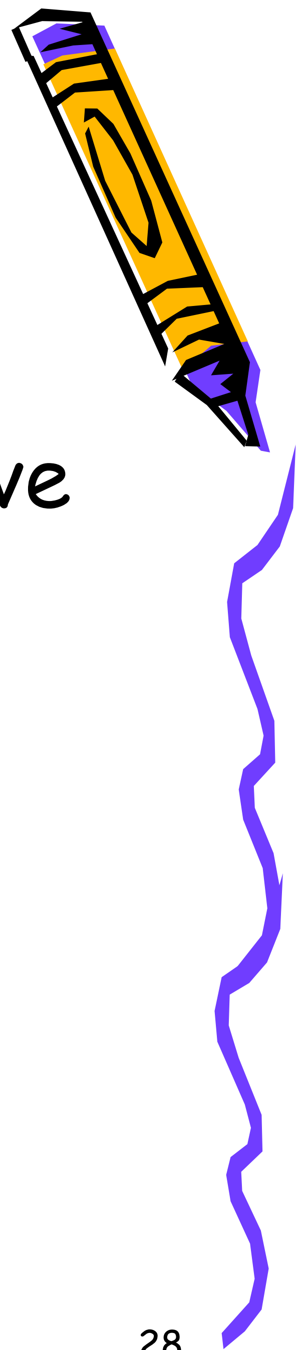
$$\mathbf{v}''' = M\mathbf{v}$$

where $M = TRS$

Deformations

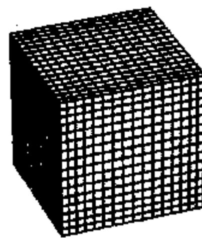
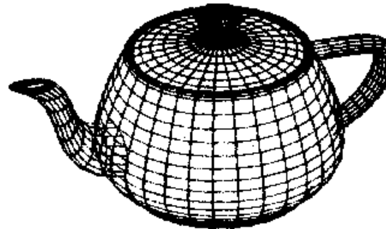
Transformations that do not preserve shape

- Non-uniform scaling
- Shearing
- Tapering
- Twisting
- Bending

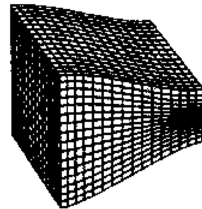
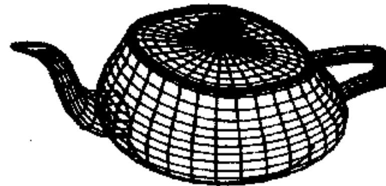


Tapering

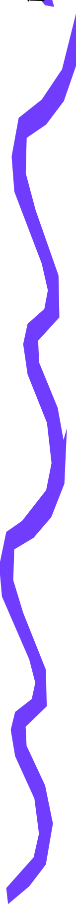
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & f(x) & 0 & 0 \\ 0 & 0 & f(x) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Original objects

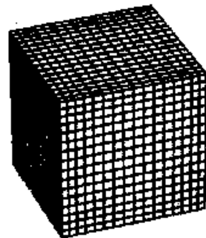
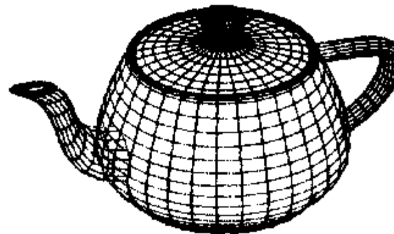



Tapering

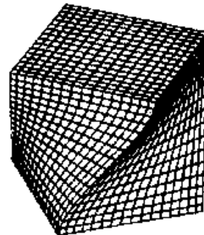
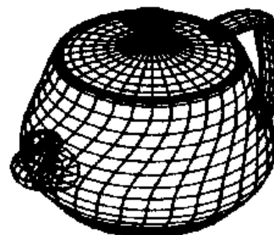


Twisting

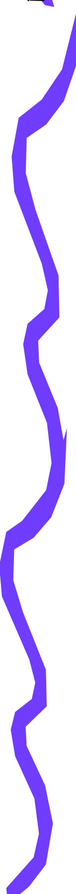
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\theta(y)) & 0 & \sin(\theta(y)) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\theta(y)) & 0 & \cos(\theta(y)) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Original objects

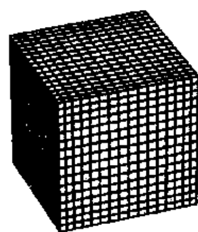



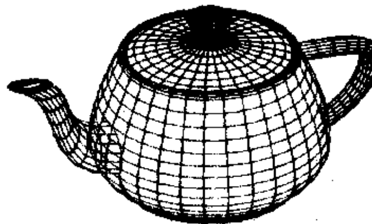
Twisting



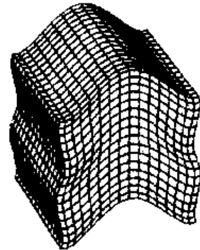
Bending

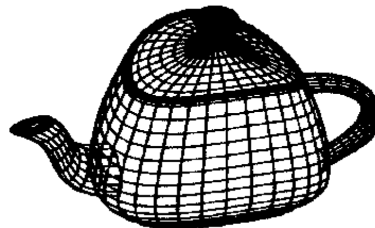
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & f(y) & g(y) & 0 \\ 0 & h(y) & k(y) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



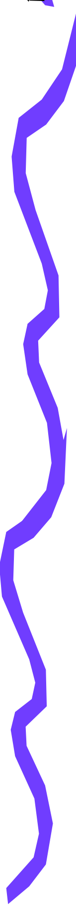


Original objects

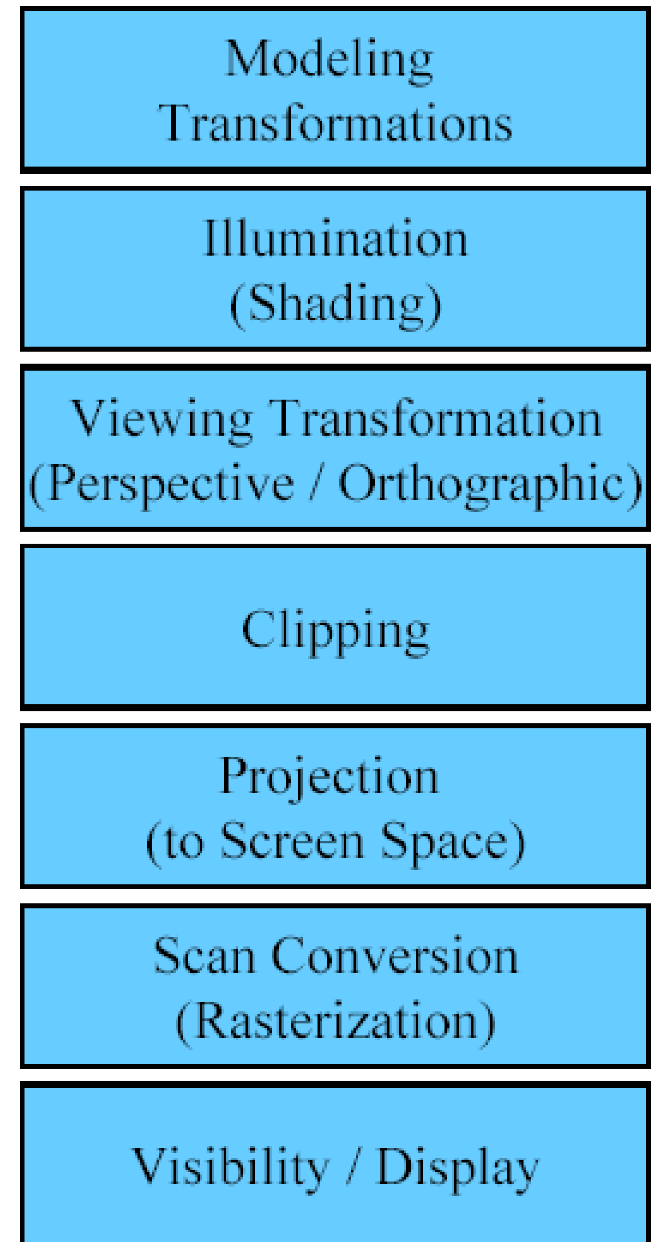




Bending

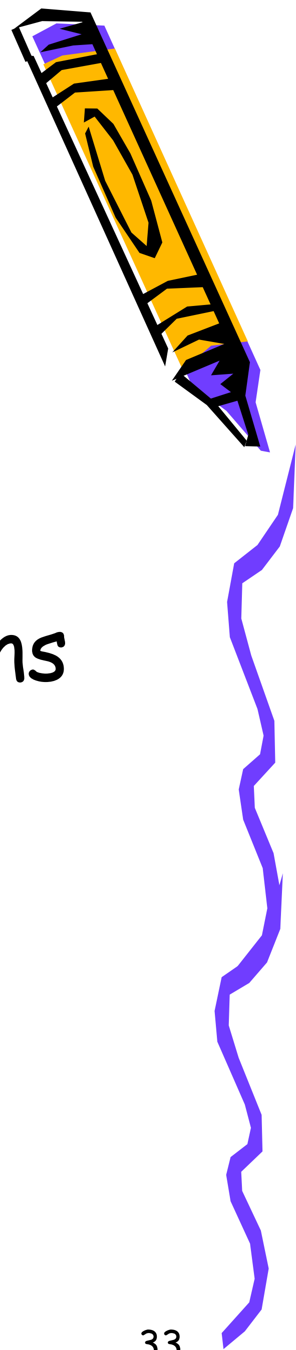


Graphics Pipeline



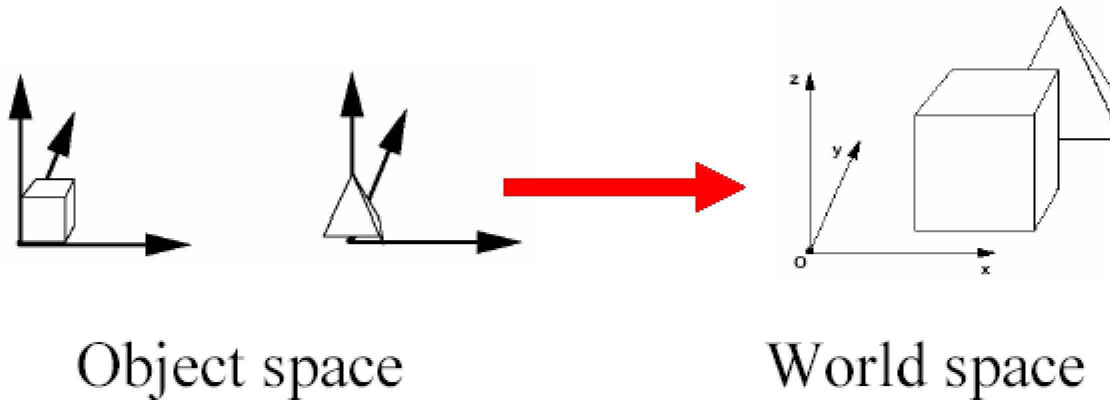
Graphics Pipeline

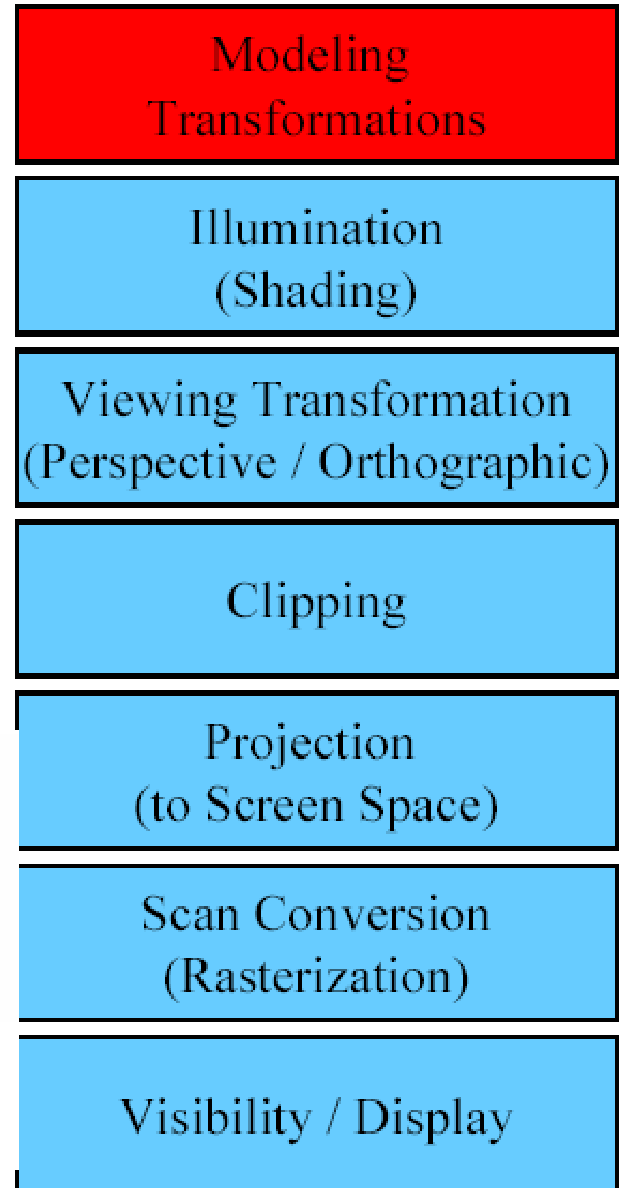
- Almost every step in the graphics pipeline involves a change of coordinate system. Transformations are central to understanding 3D computer graphics.



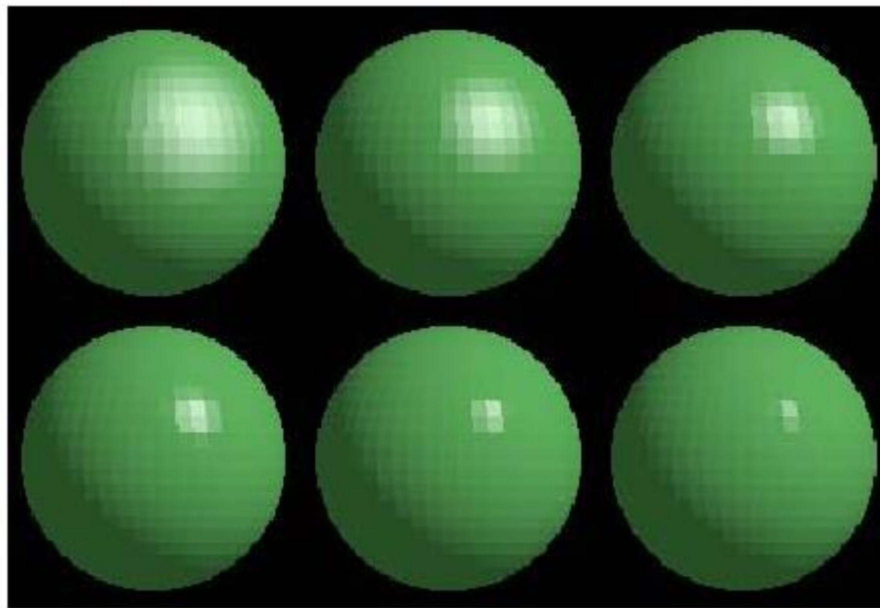
Graphics Pipeline

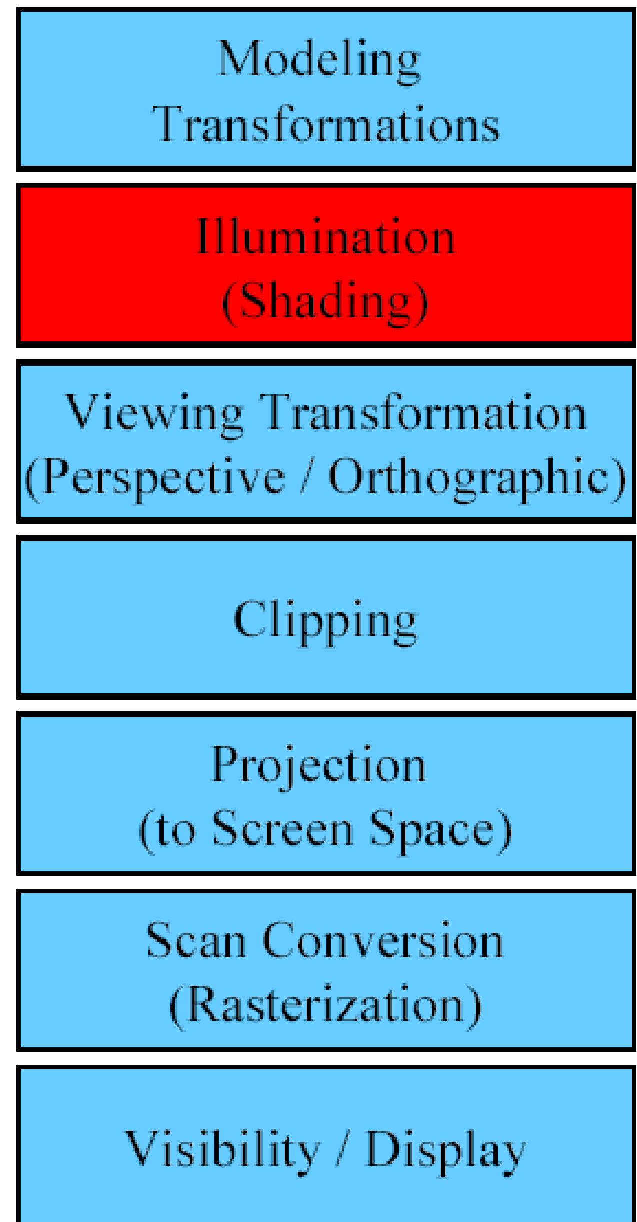
- Modeling transforms orient the models within a common coordinate frame (world space)





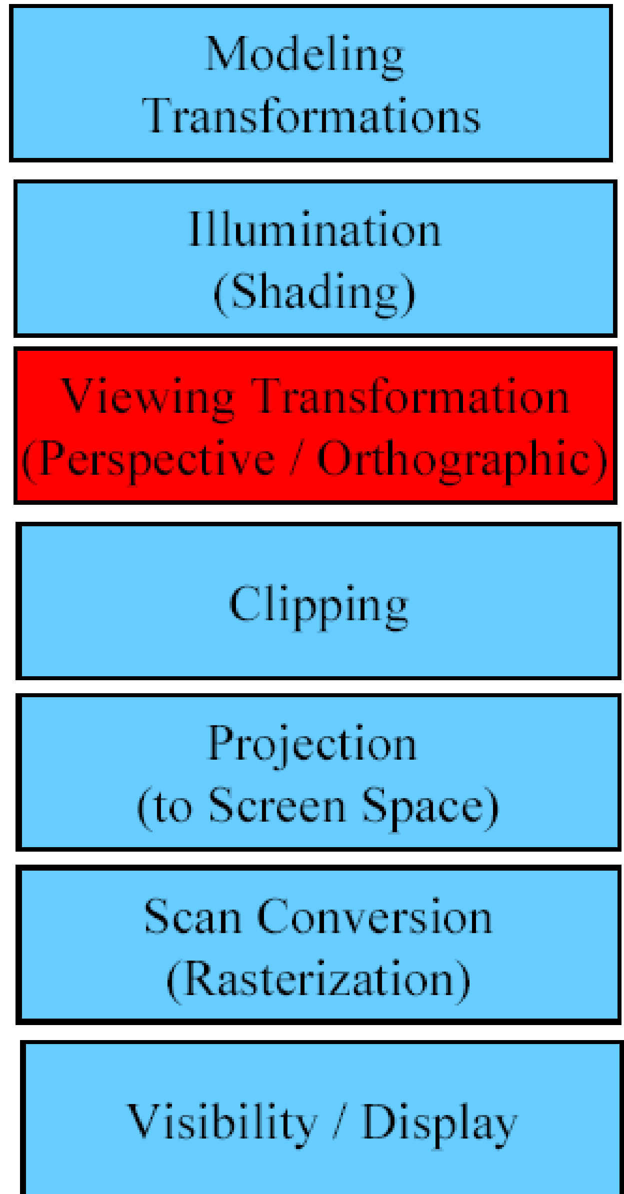
Graphics Pipeline





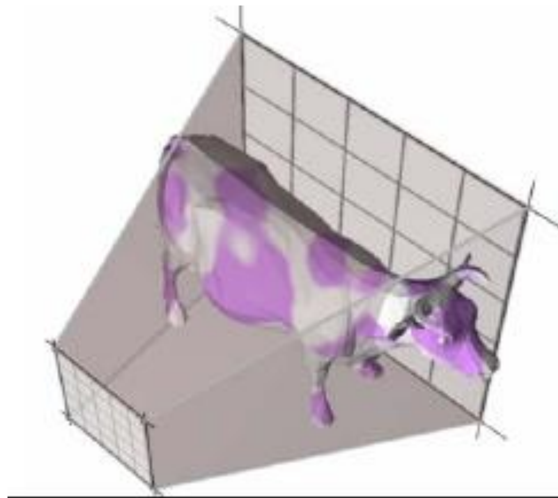
Graphics Pipeline

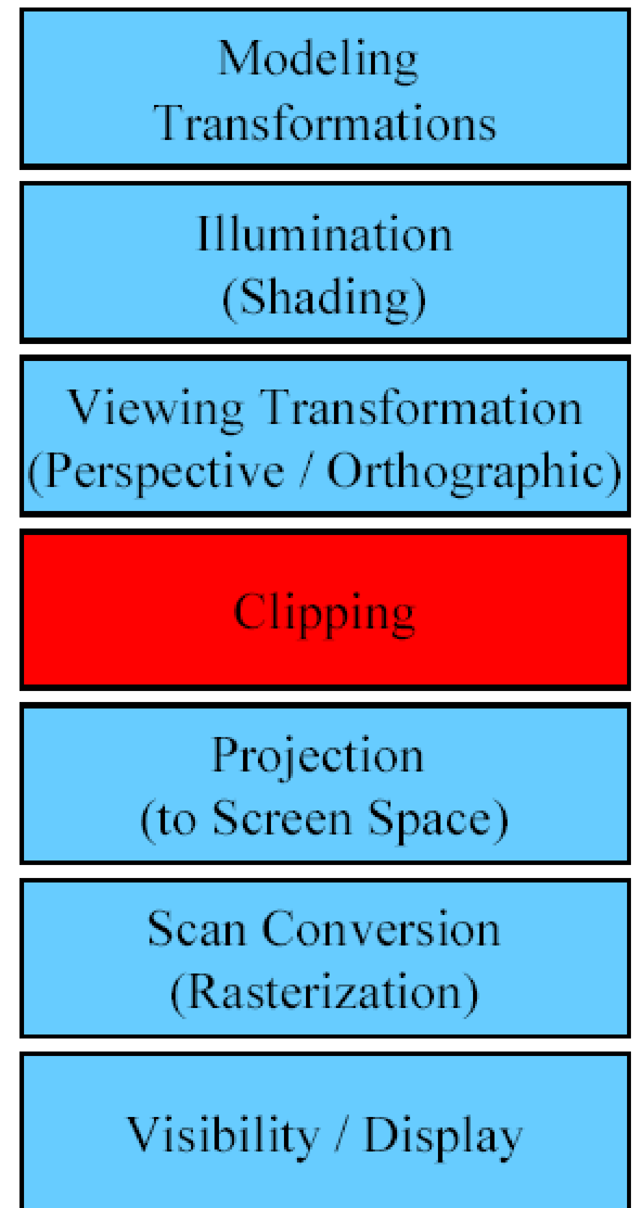
- Maps world space to eye space
- Viewing position is transformed to origin & direction is oriented along some axis (usually *z*)



Graphics Pipeline

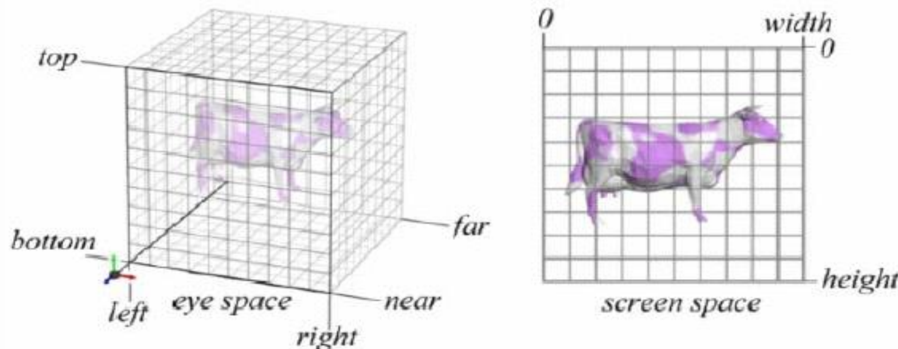
- Transform to Normalized Device Coordinates (NDC)
- Portions of the object outside the view volume (view frustum) are removed

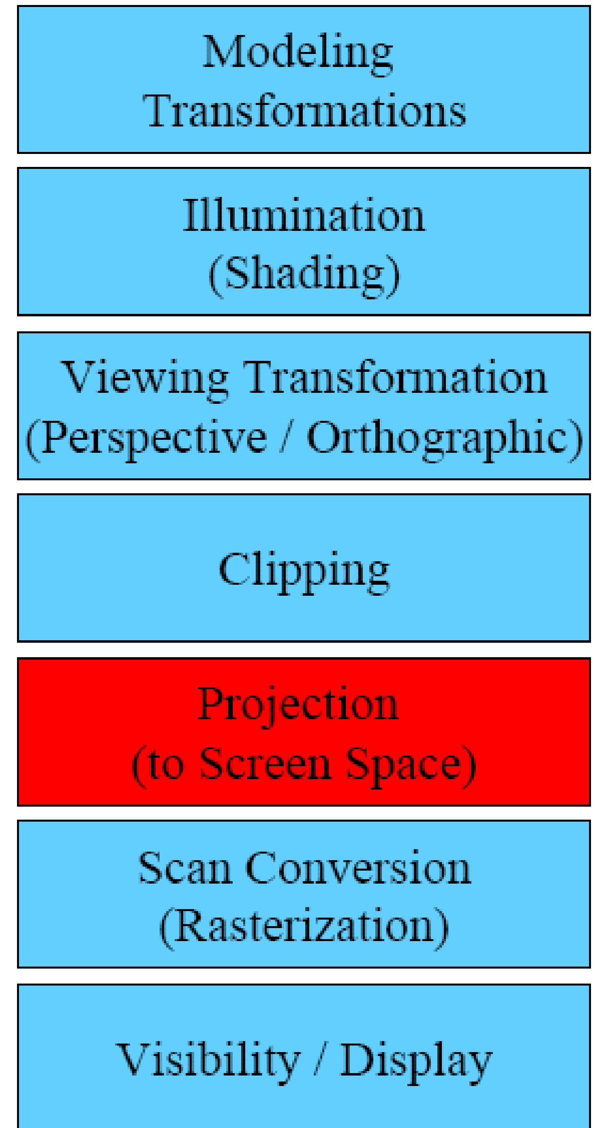




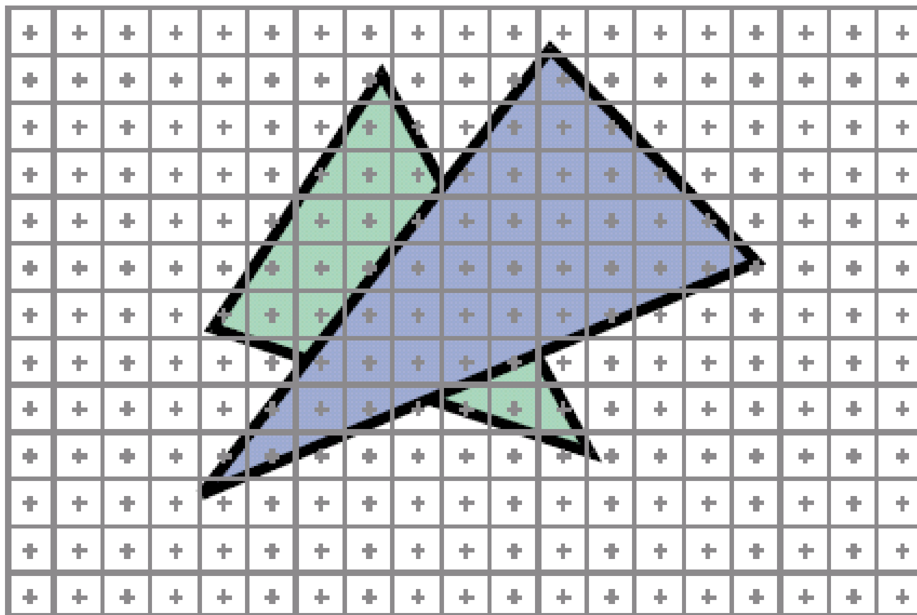
Graphics Pipeline

- The objects are projected to the 2D image place (screen space)





Graphics Pipeline



Modeling Transformations

Illumination (Shading)

Viewing Transformation (Perspective / Orthographic)

Clipping

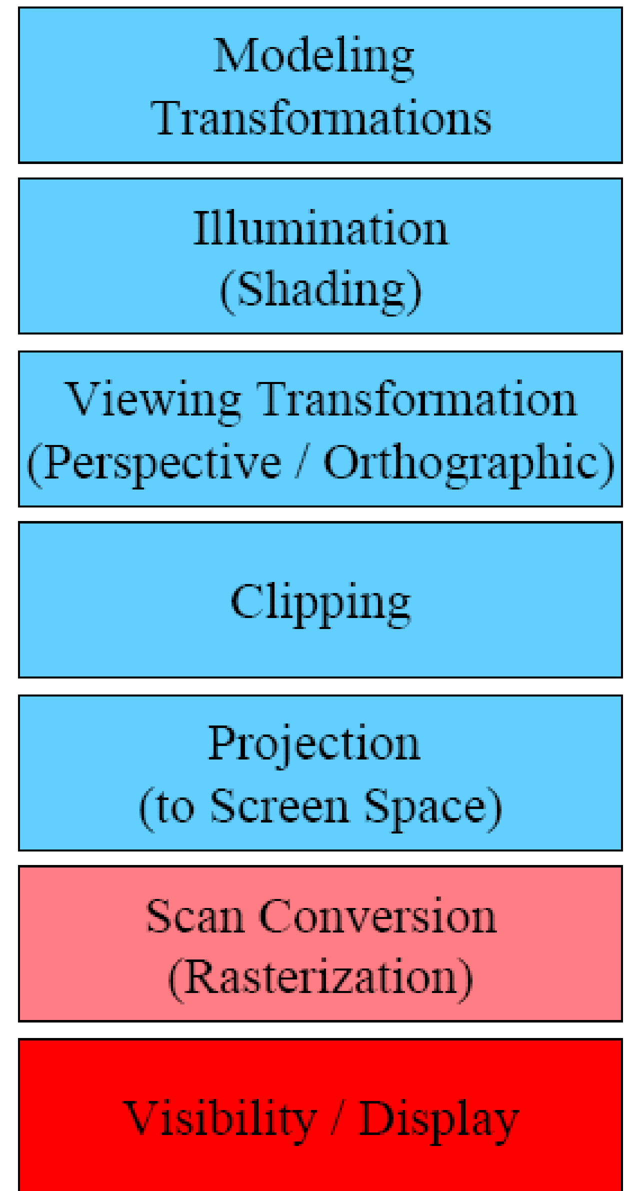
Projection (to Screen Space)

Scan Conversion (Rasterization)

Visibility / Display

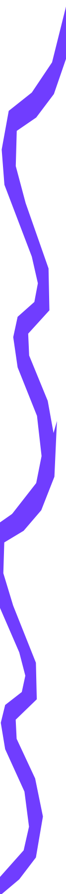
Graphics Pipeline

- Z-buffer - Each pixel remembers the closest object (depth buffer)



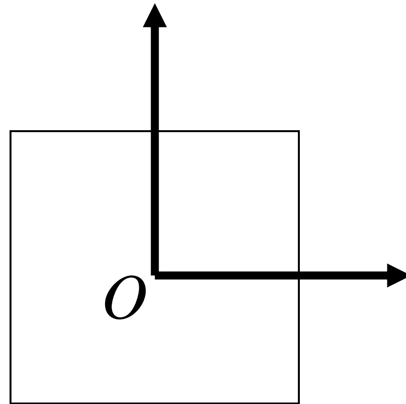
Coordinate Systems

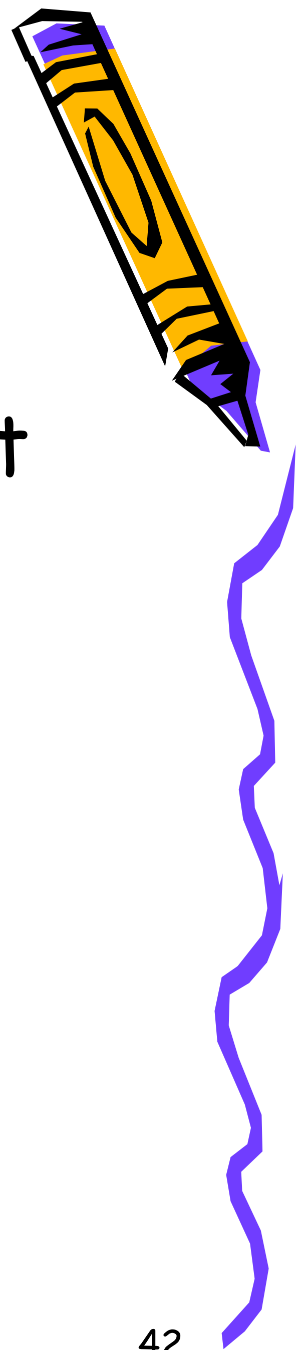
- Object coordinates
- World coordinates
- Camera coordinates
- Normalized device coordinates
- Window coordinates



Object Coordinates

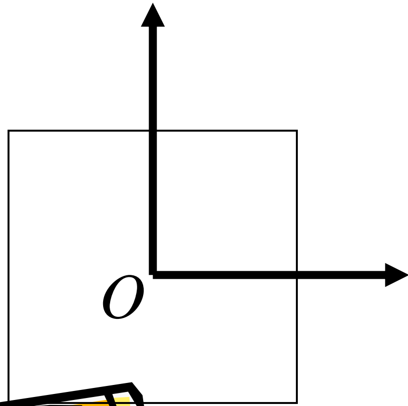
Convenient place to model the object

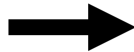


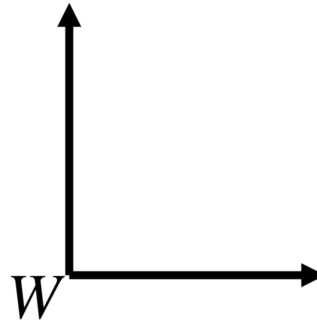


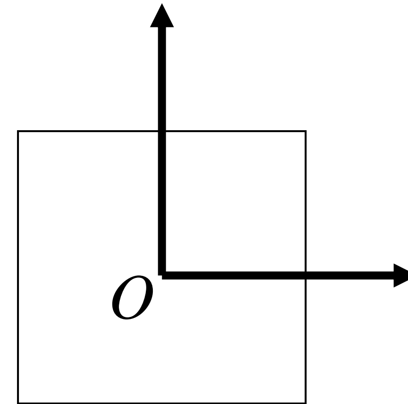
World Coordinates

Common coordinates for the scene

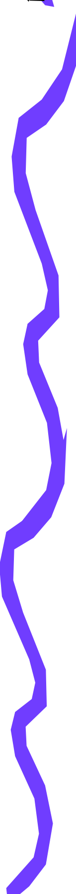




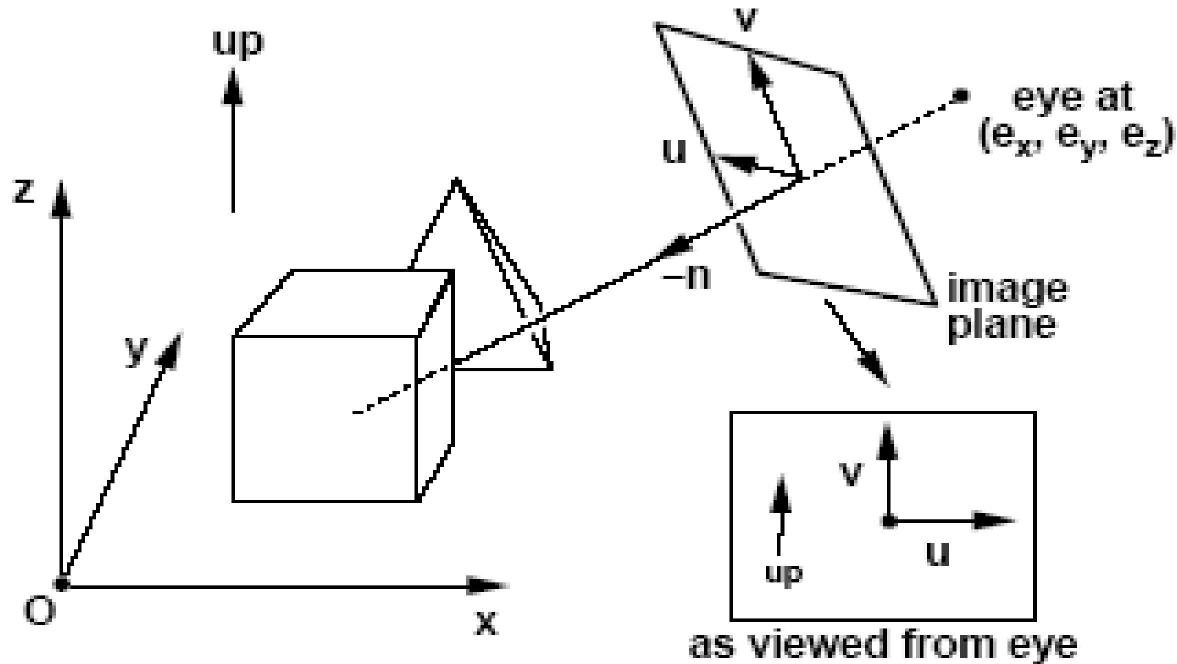




$$M_{wo} = TSR$$

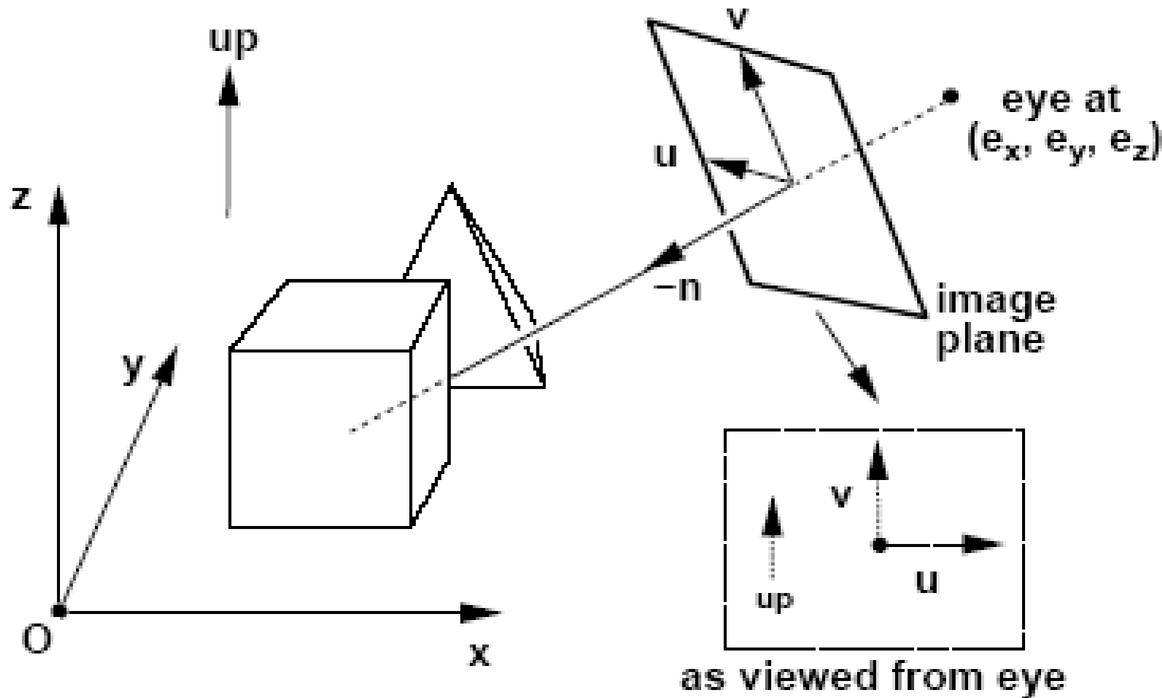


Positioning Synthetic Camera



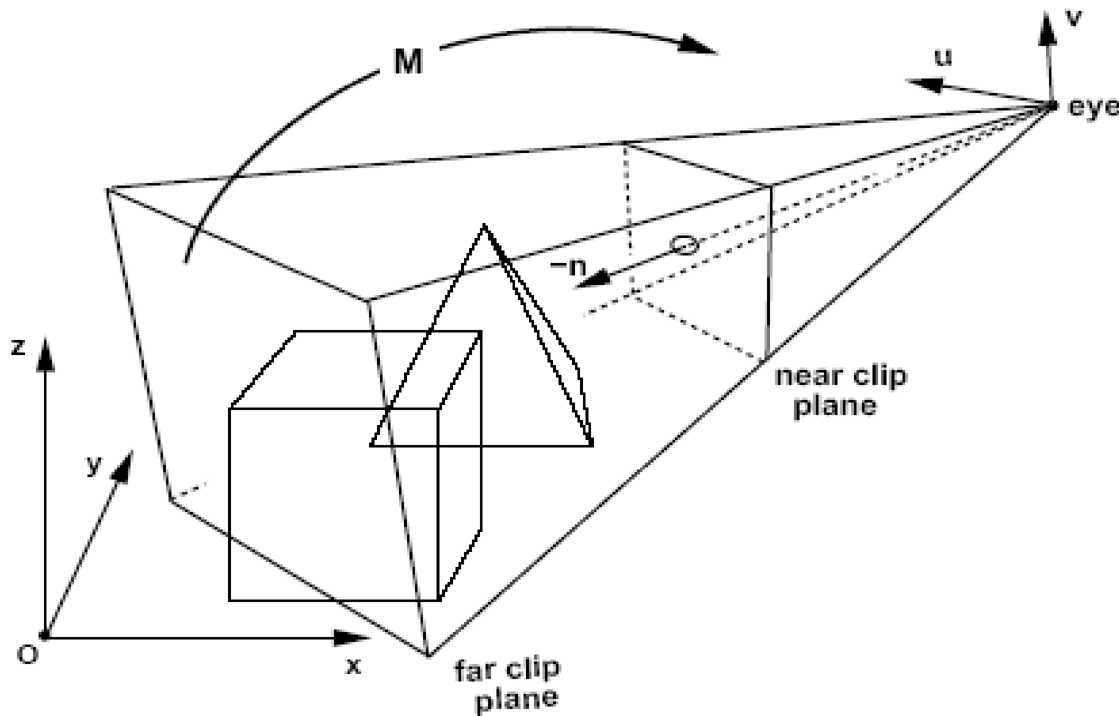
What are our "degrees of freedom" in camera positioning?
To achieve effective visual simulation, we want:
1) the eye point to be in proximity of modeled scene
2) the view to be directed toward region of interest, and
3) the image plane to have a reasonable "twist"

Eye Coordinates



Eyepoint at origin
u axis toward "right" of image plane
v axis toward "top" of image plane
view direction along *negative* **n** axis

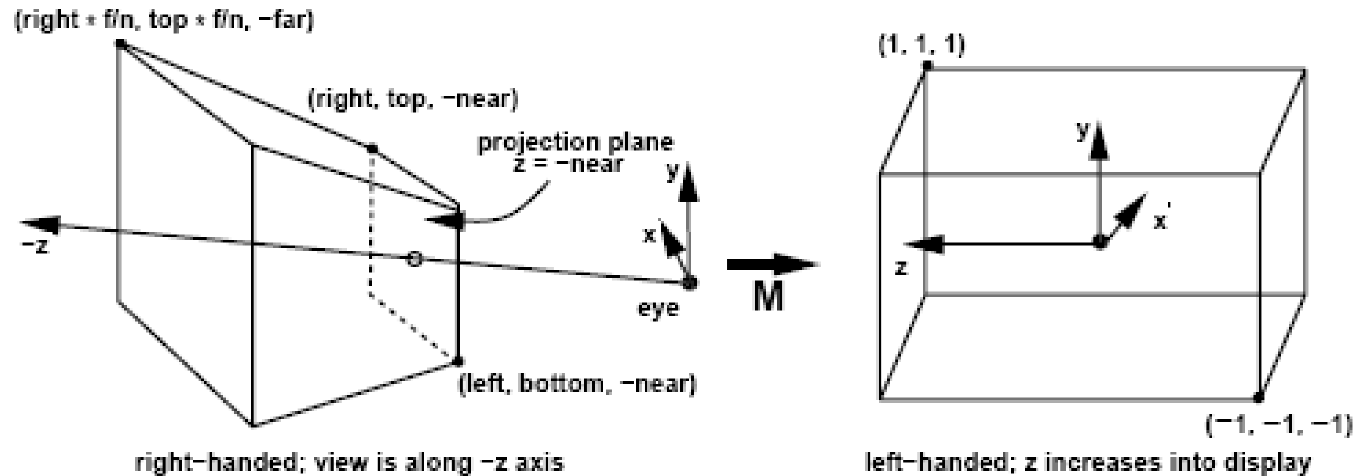
Transformation to Eye Coordinates



Our task: construct the transformation **M** that re-expresses world coordinates in the viewer frame

Where are we?

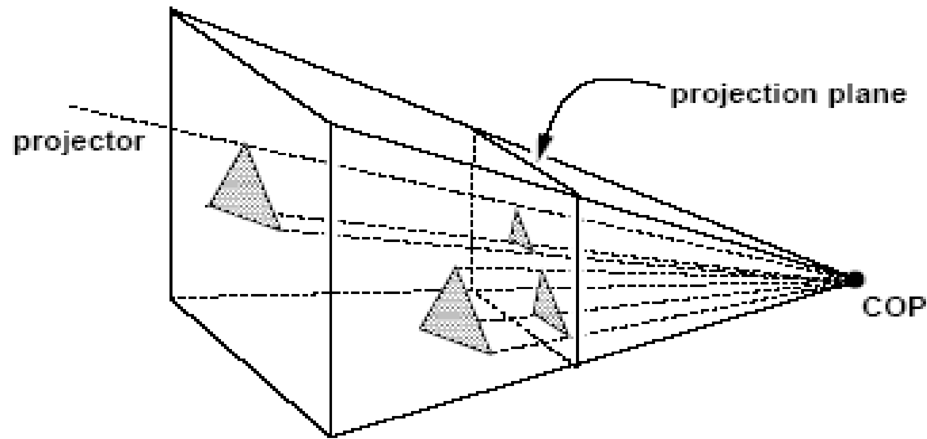
We've re-expressed world geometry in eye's frame of reference:

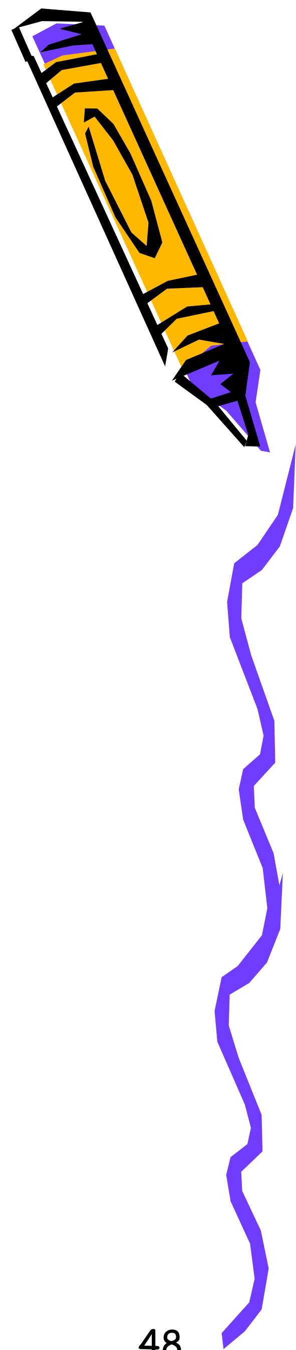


Next we must transform to NDC (Normalized Device Coordinates)
to prepare for (simple) clipping and projection

For that, we need the *Perspective Transformation*
We'll study *Perspective Projection* first, then generalize

What is Projection?



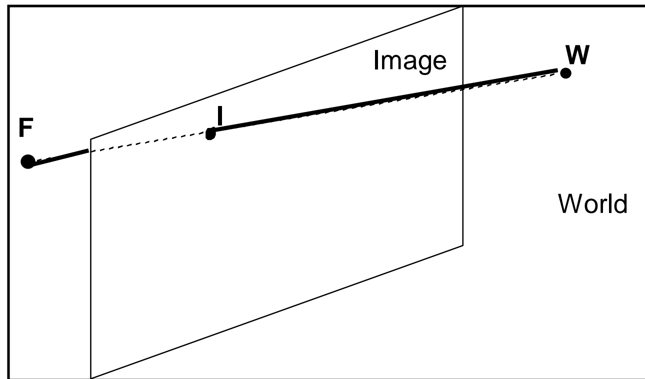


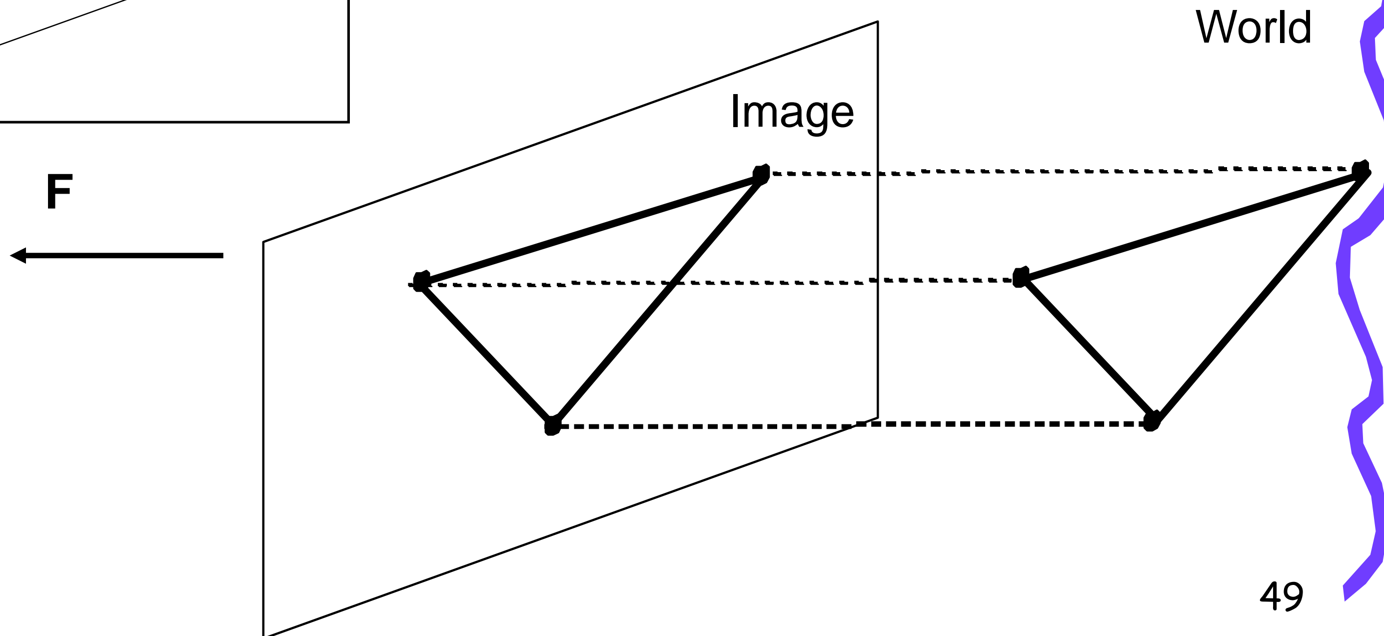
Any operation that reduces dimension (e.g., 3D to 2D)

Orthographic Projection
Perspective Projection

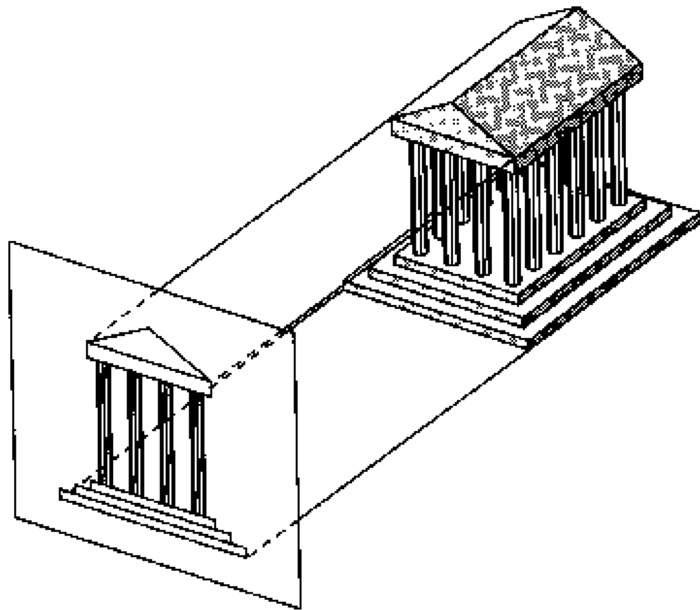
Orthographic Projection

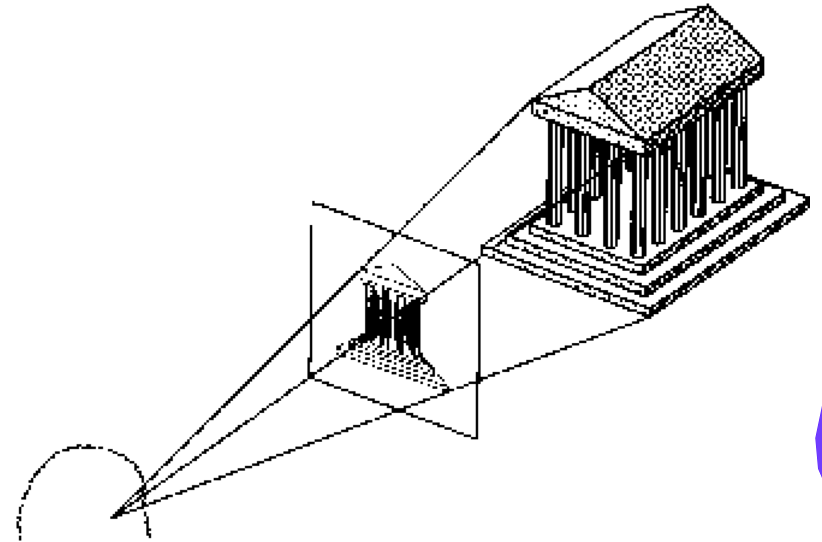
- focal point at infinity
- rays are parallel and orthogonal to the image plane



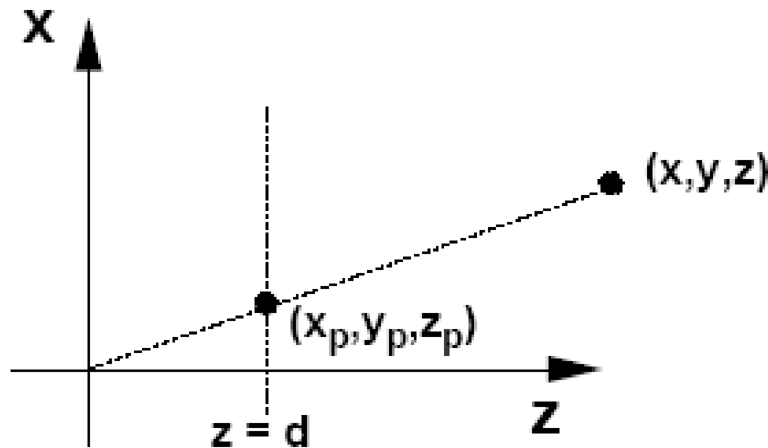


Comparison





Perspective Projection



What are coordinates of projected point x_p, y_p, z_p?

By similar triangles,

$$\frac{x_p}{d} = \frac{x}{z} \qquad \frac{y_p}{d} = \frac{y}{z}$$

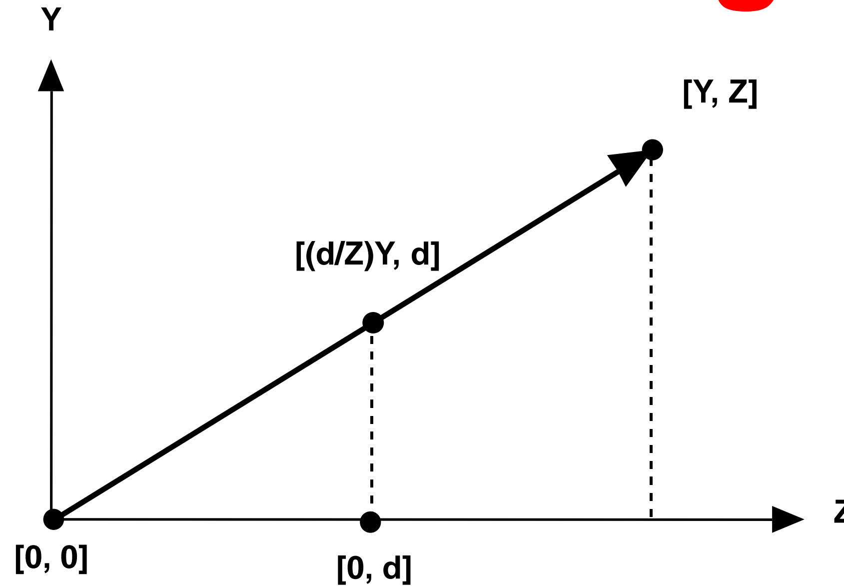
Multiplying through by d yields

$$x_p = \frac{d \cdot x}{z} = \frac{x}{z/d} \qquad y_p = \frac{d \cdot y}{z} = \frac{y}{z/d} \qquad z_p = d$$

$z = 0$ not allowed (what happens to points on plane $z = 0$?)

Operation well-defined for all other points

Similar Triangles



- Similar situation with *x*-coordinate
- Similar Triangles:

  point [x,y,z] projects to [(d/z)x, (d/z)y, d]

Projection Matrix

Projection using homogeneous coordinates:

– transform [x, y, z] to [(d/z)x, (d/z)y, d]

$$\begin{bmatrix} d & 0 & 0 & 0 \\ 0 & d & 0 & 0 \\ 0 & 0 & d & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} dx & dy & dz & z \end{bmatrix} \Rightarrow \begin{bmatrix} \frac{d}{z}x & \frac{d}{z}y & \mathrm{d} \end{bmatrix}$$

Divide by 4th coordinate (the "w" coordinate)

- 2-D image point:
  - discard third coordinate
  - apply viewport transformation to obtain physical pixel coordinates

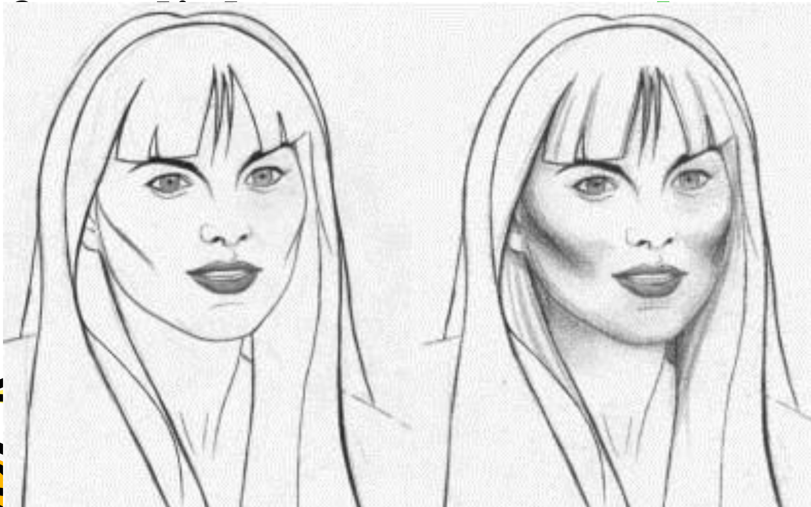
Shading

[Drawing] Shading is a process used in drawing for depicting levels of darkness on paper by applying media more densely or with a darker shade for darker areas, and less densely or with a lighter shade for lighter areas.

[Computer graphics] Shading refers to the process of altering a color based on its angle to lights and its distance

ealistic effect. Shading is

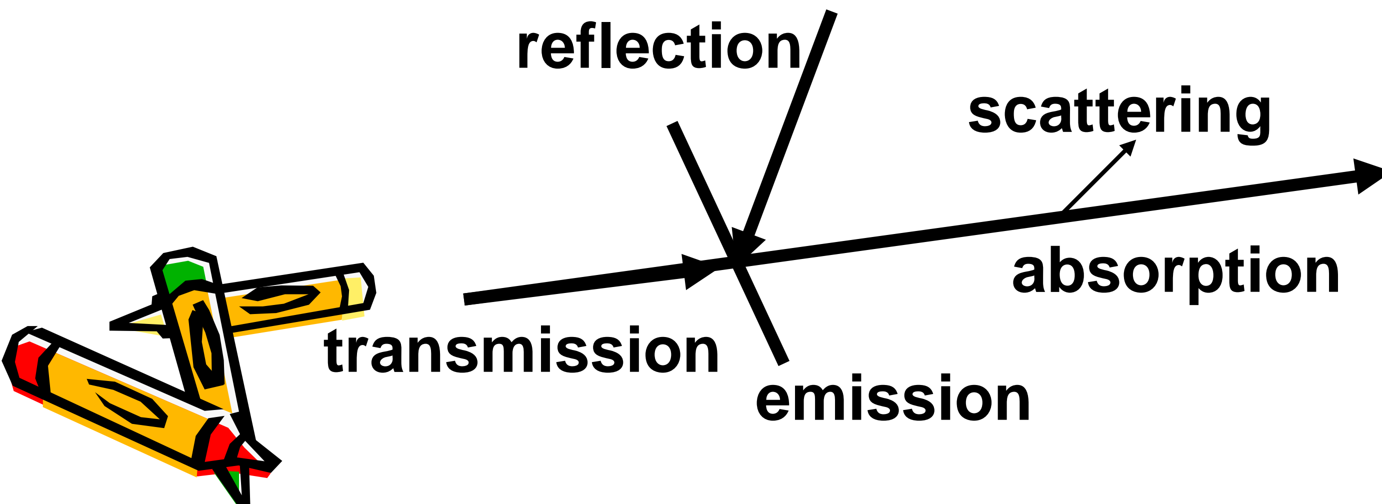
ng



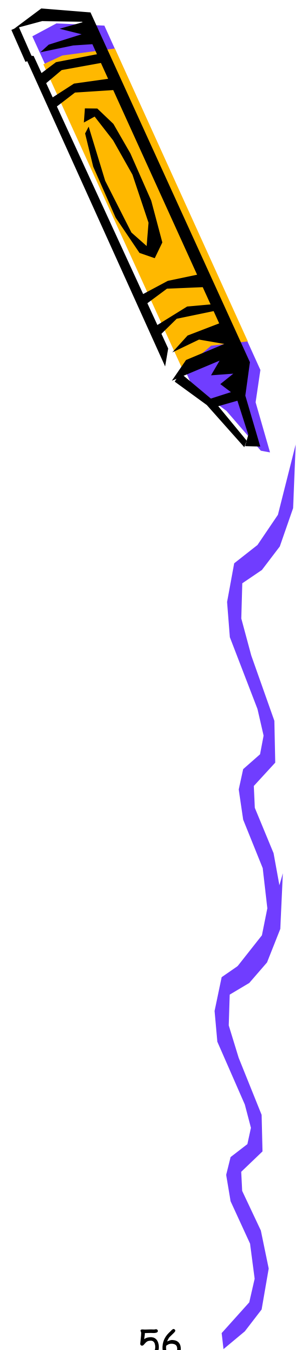
Shading

Light comes from many sources:

$$light = emitted + reflected + transmitted - scattered - absorbed$$

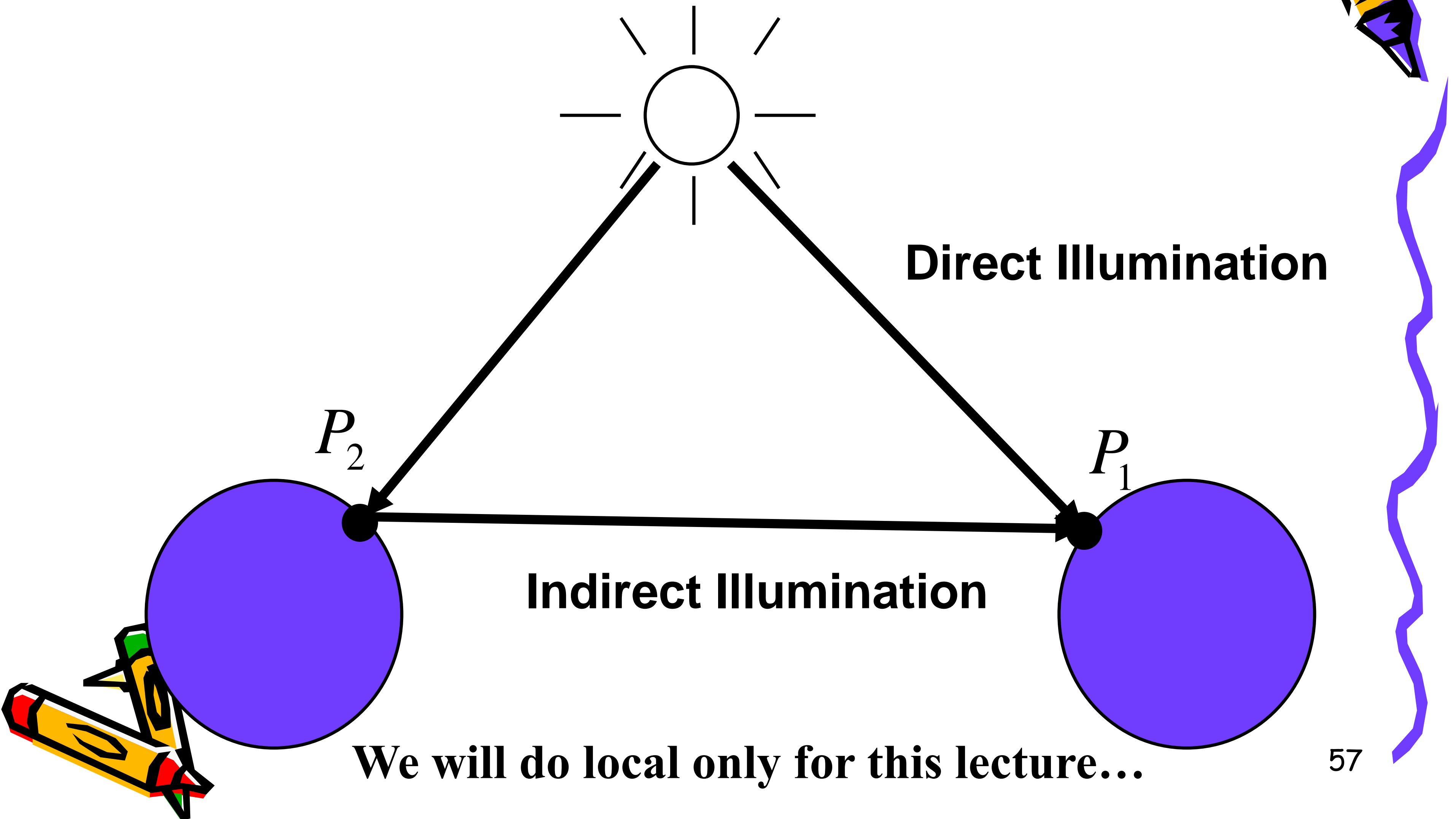


Local versus Global Illumination



- **Local Illumination**
  - **Only considers direct illumination**
  - **No reflection**
  - **No refraction**
  - **Shadows possible**
- Global Illumination
  - Considers indirect illumination
  - Reflection
  - Refraction
  - Shadows

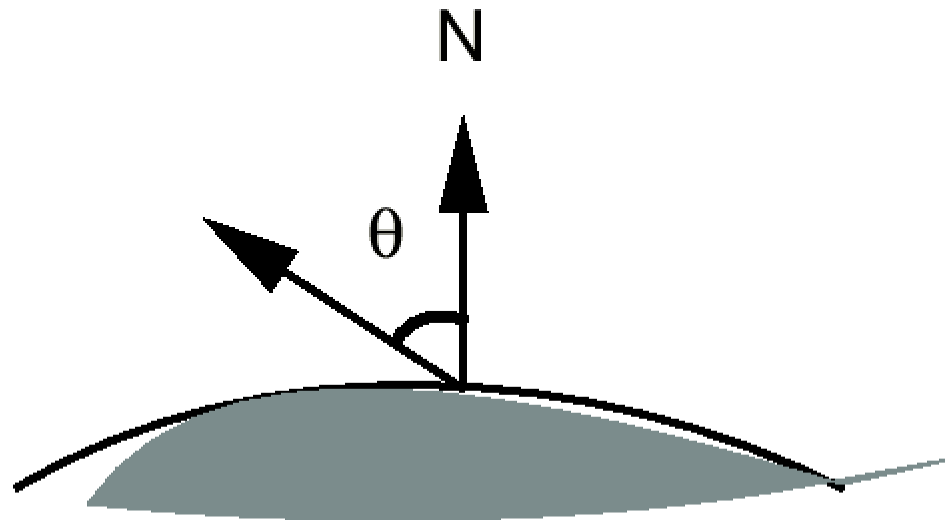
Local versus Global Illumination

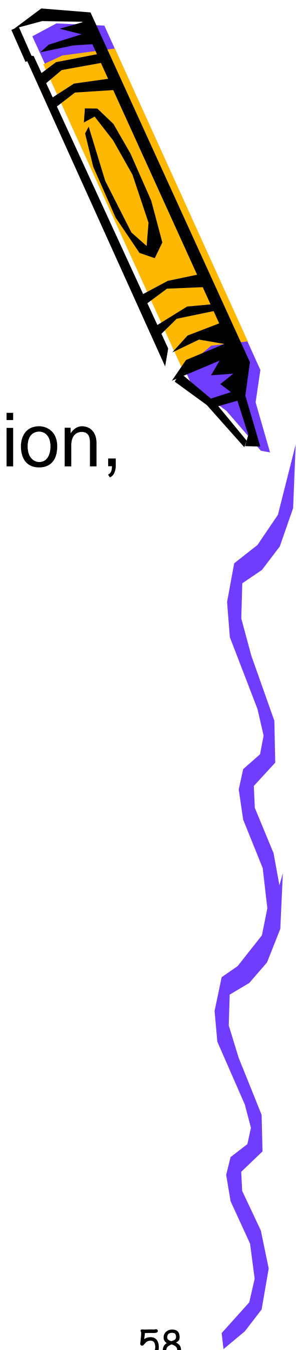


We will do local only for this lecture…

Local illumination

- Only consider the light, the observer position, and the object material properties





Local versus Global Illumination

Images courtesy of Francois Sillion

To understand shading properly, we need to review some basic notions of physics…

Phong Reflection

Assume point lights and direct illumination only

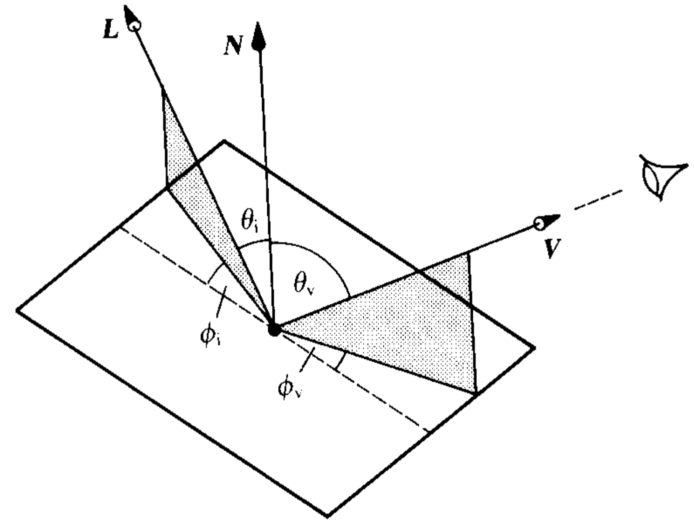
$$I = I_{ambient} + I_{diffuse} + I_{specular}$$

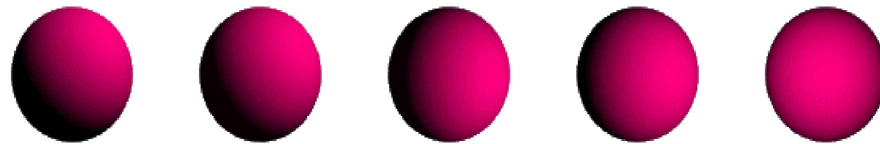
Phong Diffuse Component

Diffuse component depends only on incident angl

$$I_{diffuse} = I_l \, k_d \cos\theta$$

$$= I_l \, k_d (N \cdot L)$$



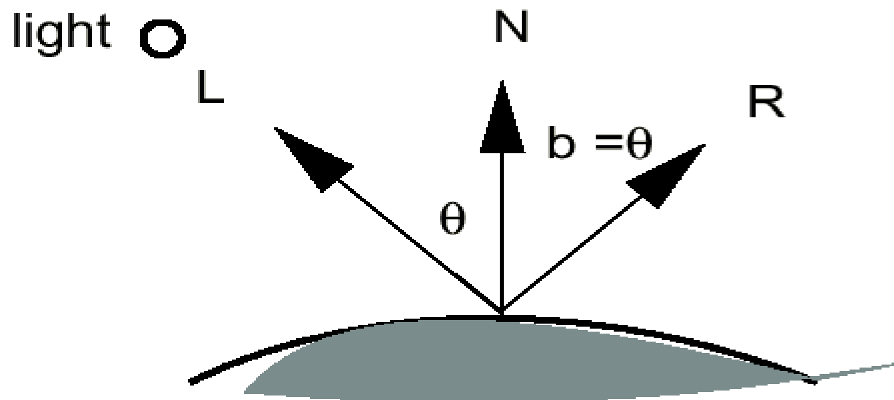


N.B: L and N are unit…

Image courtesy of Watt, 3D Computer Graphics

Specular Light

- These are the bright spots on objects (such as polished metal, apple ...)
- Light reflected from the surface unequally to all directions.
- The result of near total reflection of the incident light in a concentrated region around the specular reflection angle



Phong Reflection

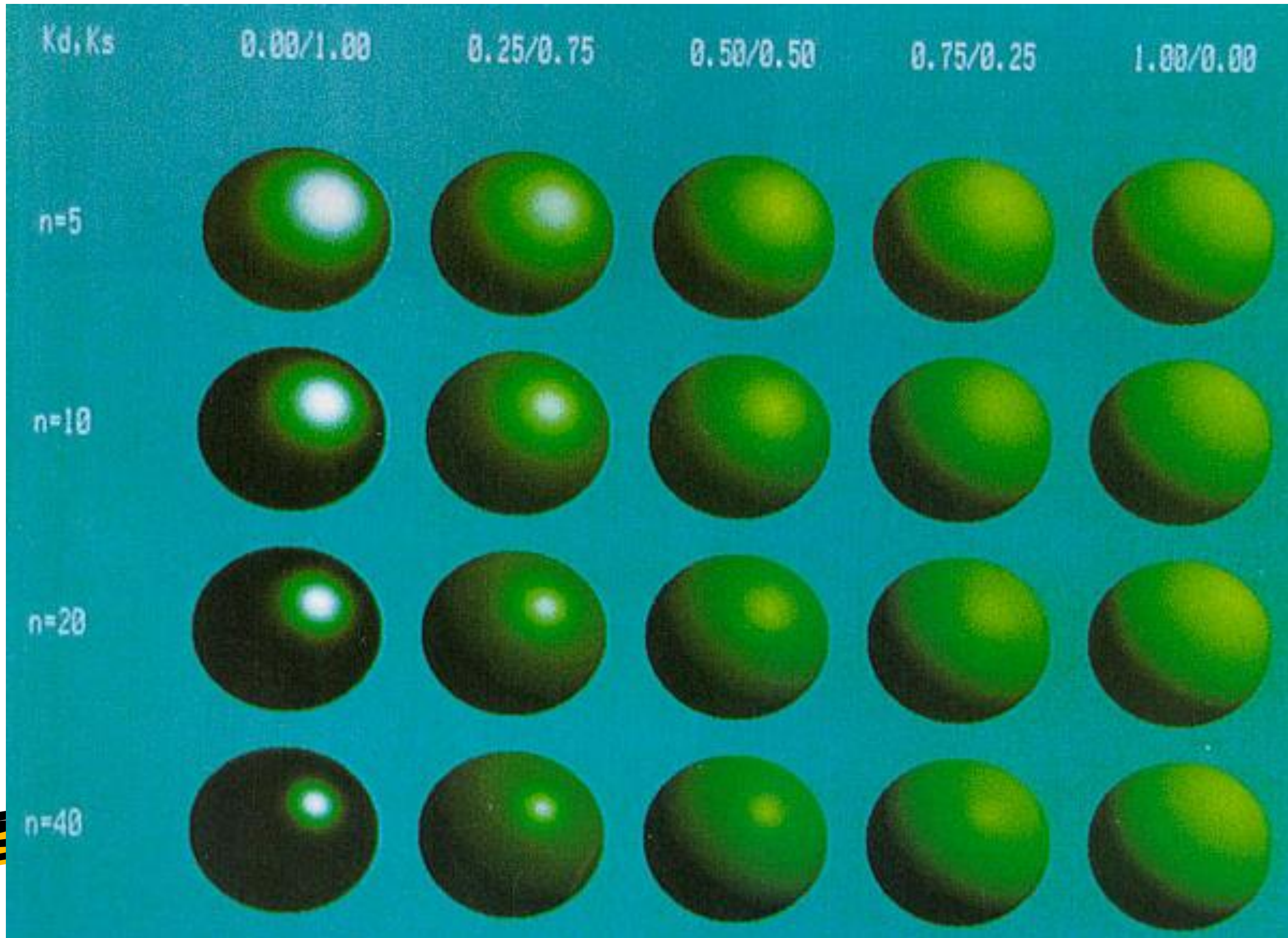
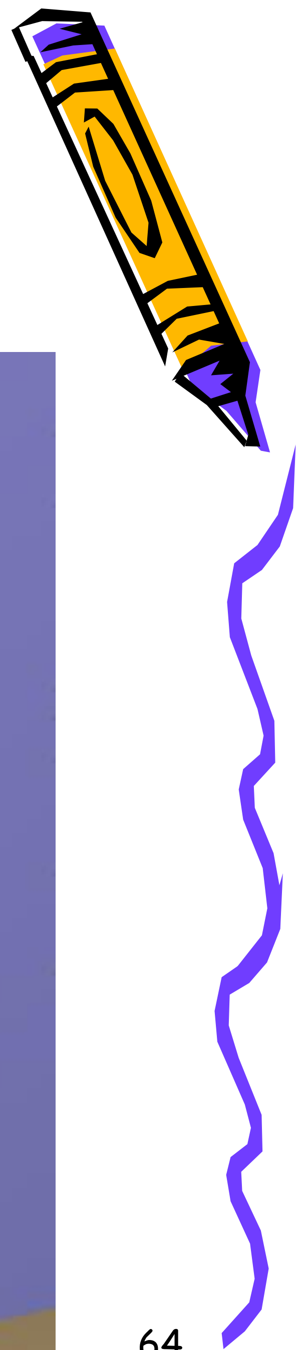
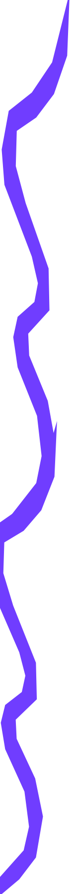


Image courtesy of Watt, 3D Computer Graphics

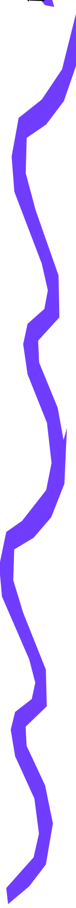
Aluminium

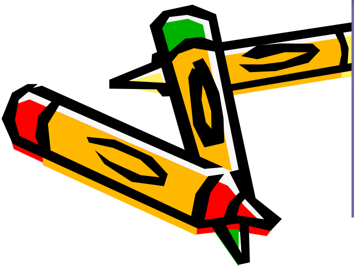


Bronze



Chrome

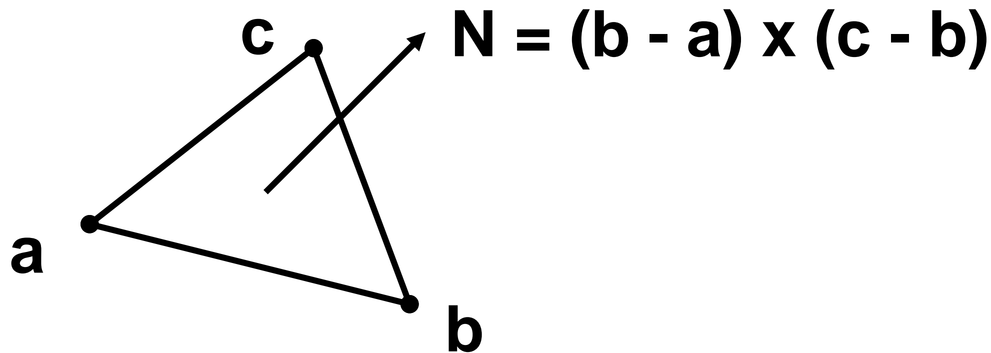




Vertex Normals vs. Face Normals

What are the normals to the surface?

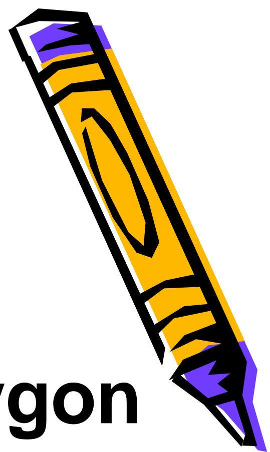
Each polygonal face has a normal.

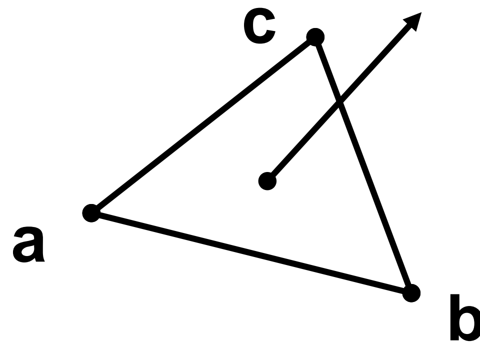


We call these face normals.

Flat Shading

Assume a constant color across the polygon

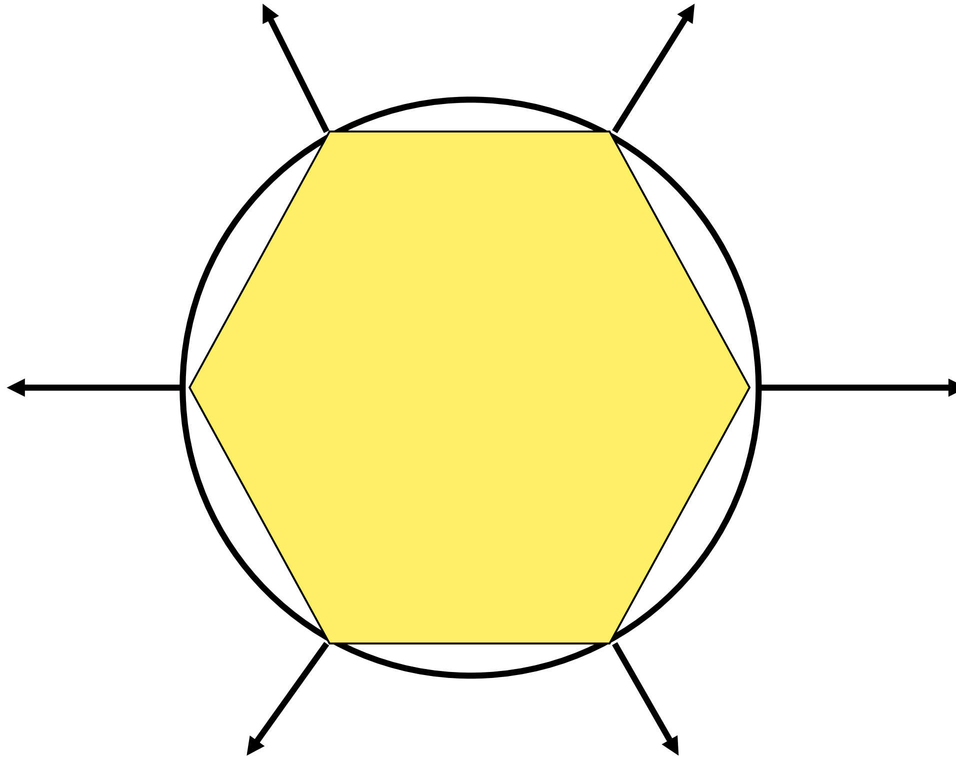




Uses face normals
Equivalent to single point sampling…
Polygon mesh is only an approximation.
Can we do better?

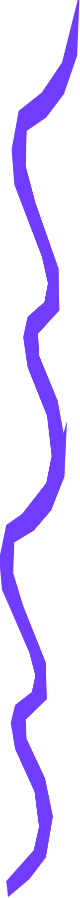
Vertex Normals vs. Face Normals

- **Should use the actual surface's normals**

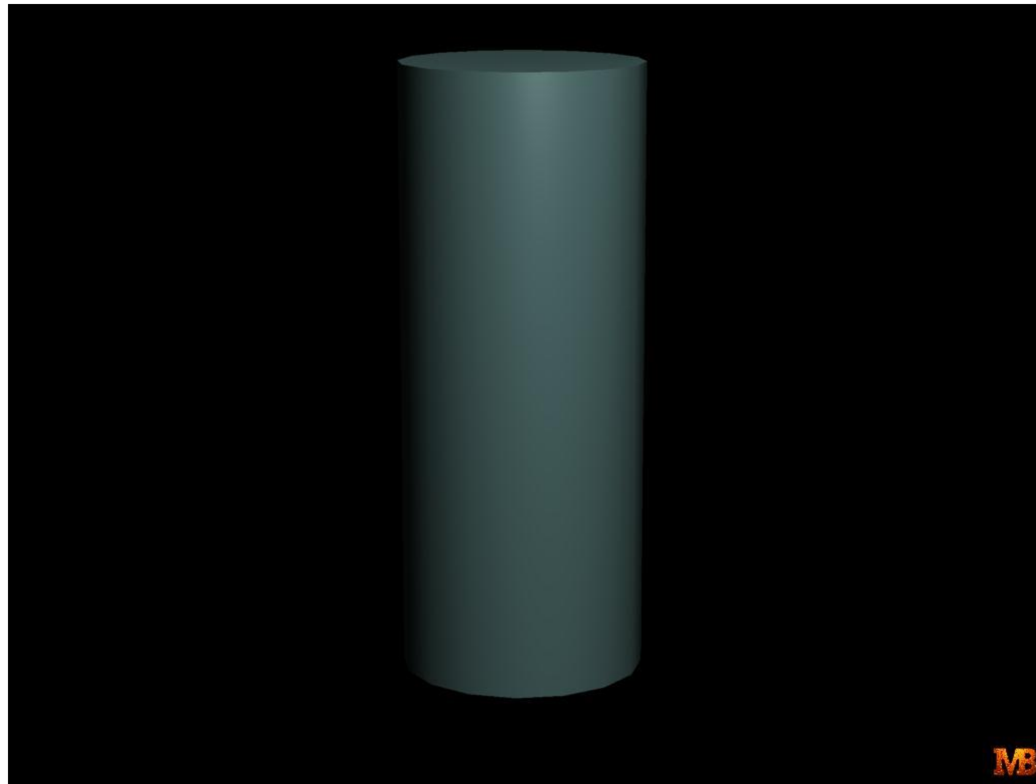


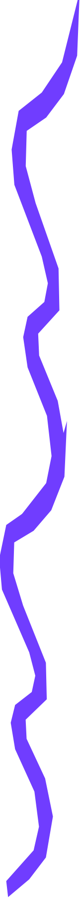
- **Usually stored at the vertices of the object**
- **Can calculate as averages of face normals**

Mach Band ?



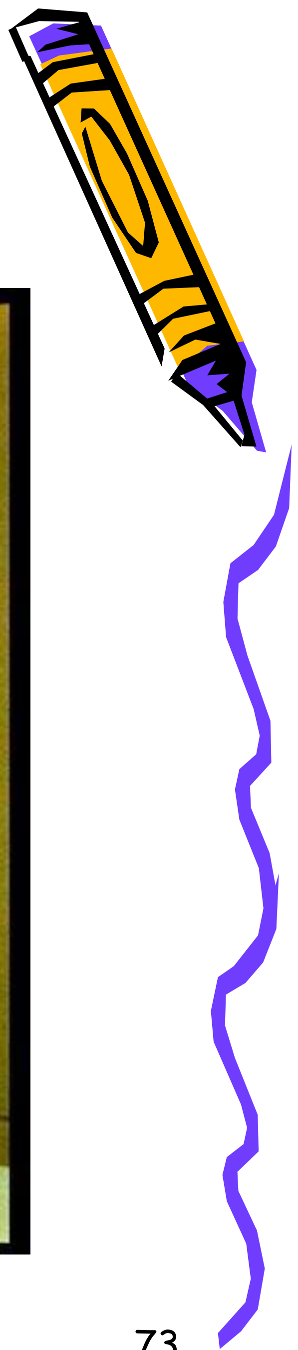
Mach Band ?



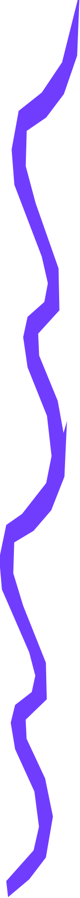


Un-lit

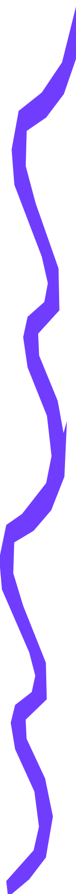
Flat Shading

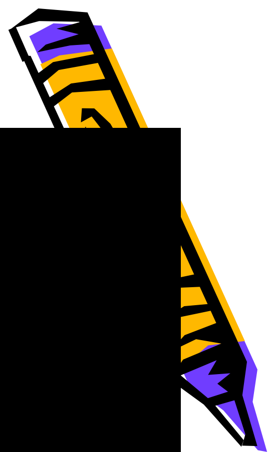
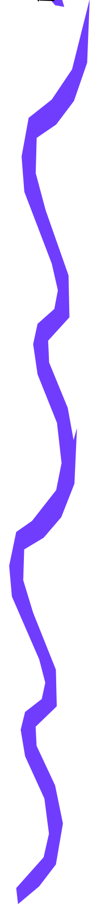


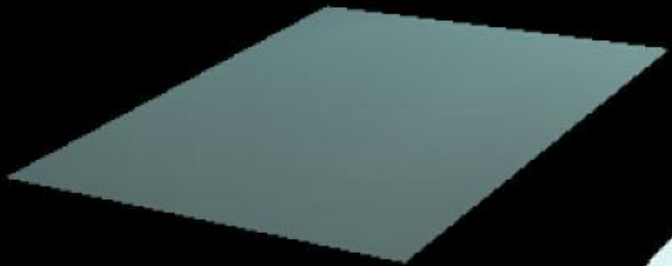
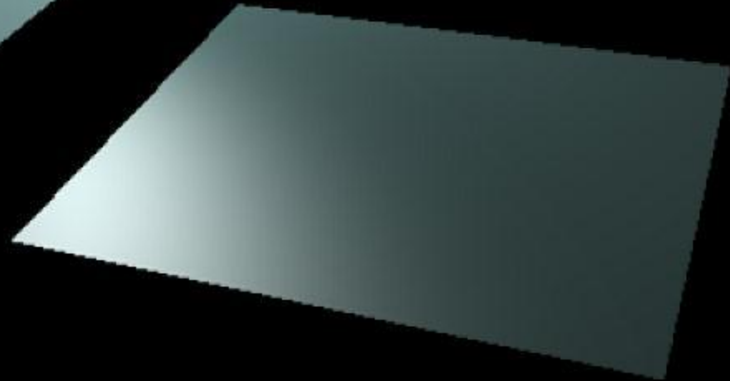
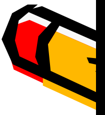
Gouraud Interpolation – Interpolated Shading



Phong Interpolation – Per pixel Shading






Gouraud
Phong

Interpolation

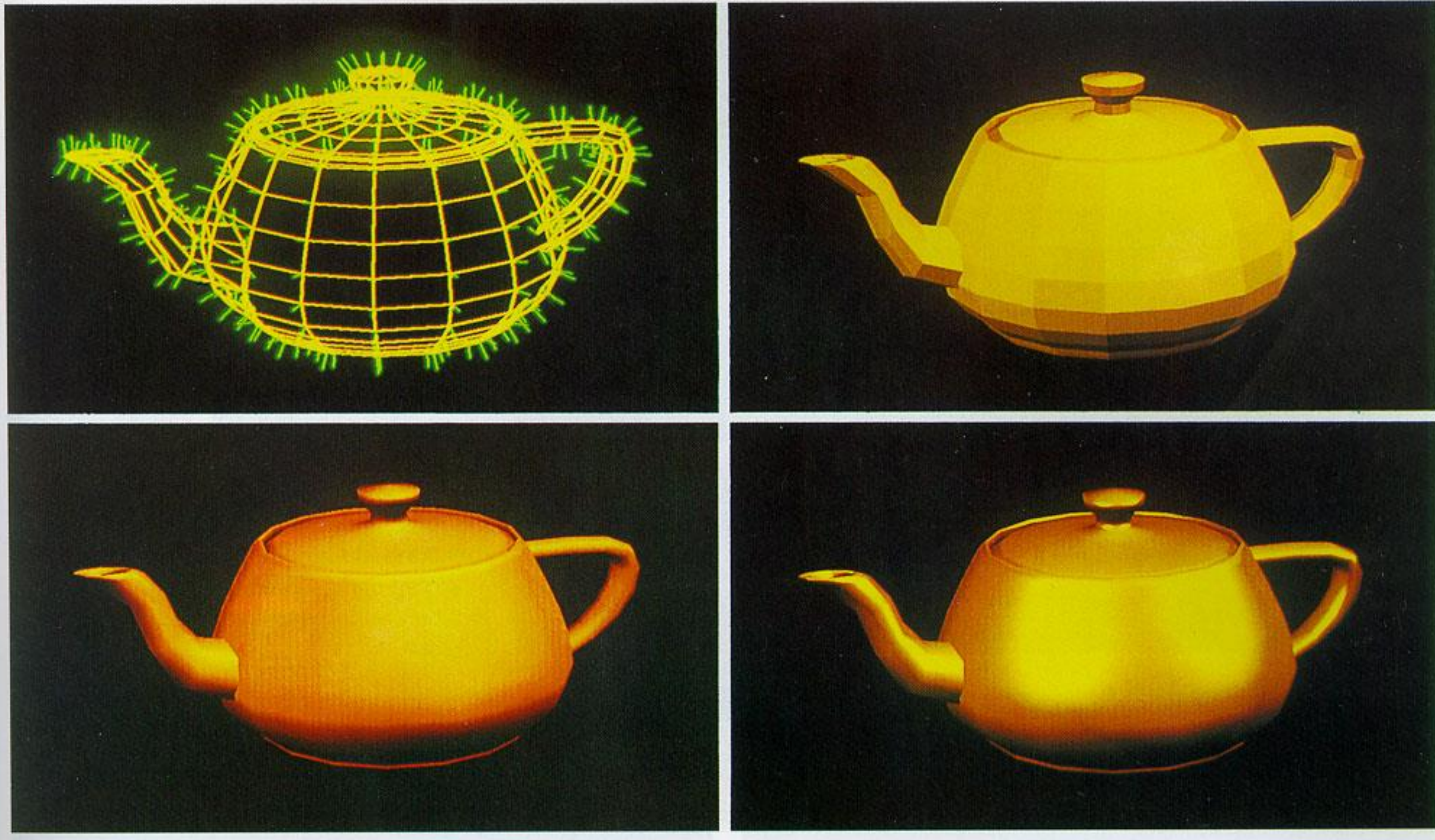
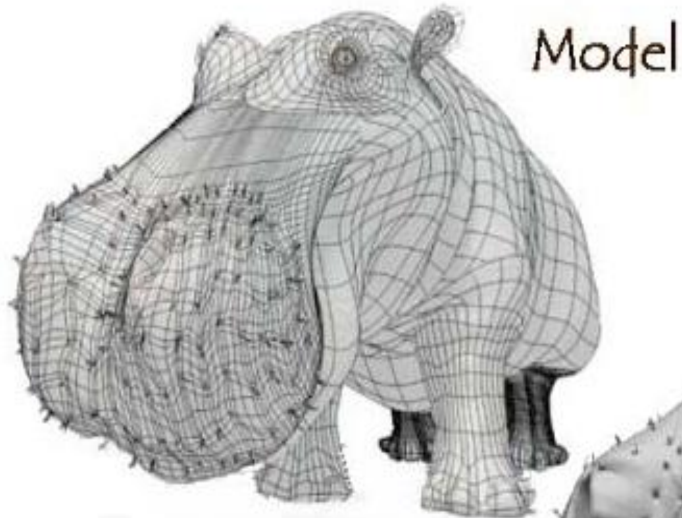
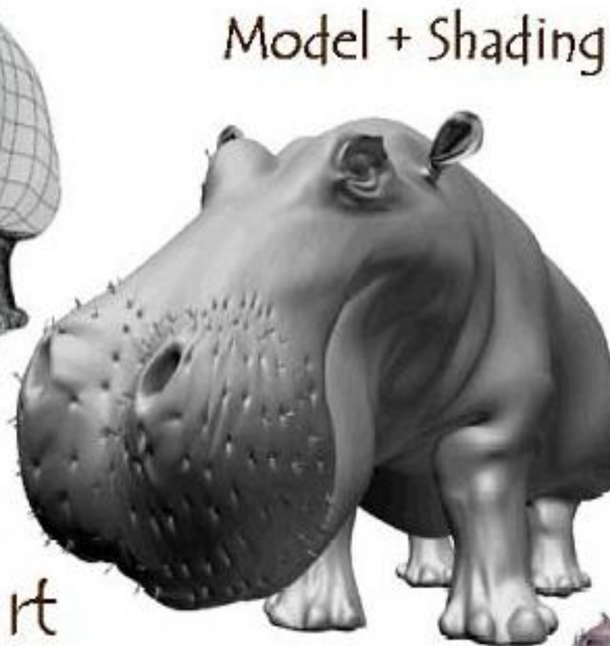


Image courtesy of Watt & Watt, Advanced Animation and Rendering Techniques

The Quest for Visual Realism



Model



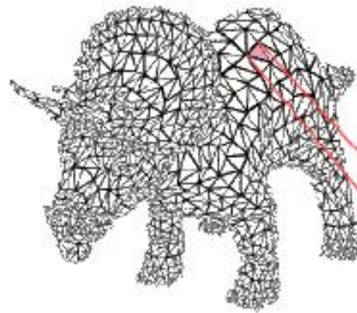
Model + Shading

Model + Shading + Textures

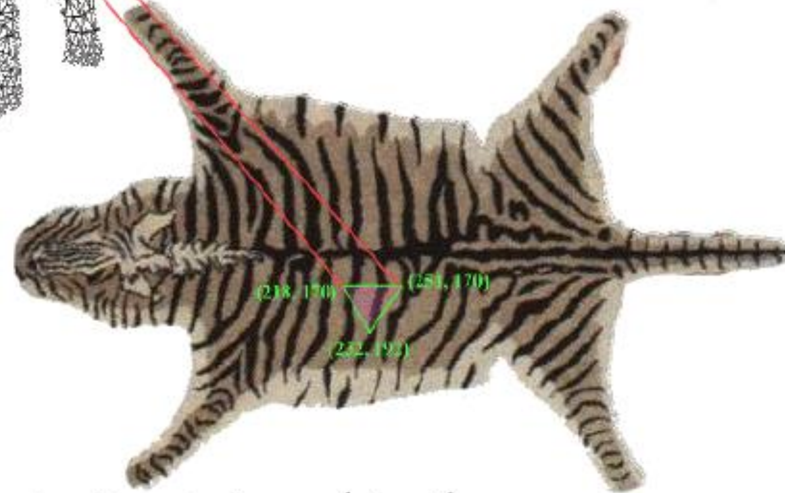
At what point do things start looking real?

Photo-textures

The concept is very simple!



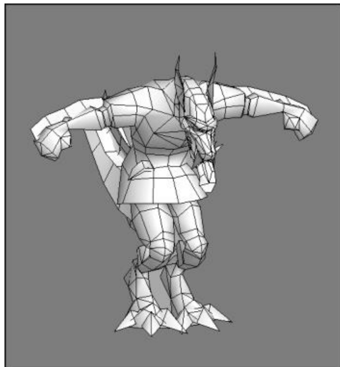
For each triangle in the model establish a corresponding region in the phototexture

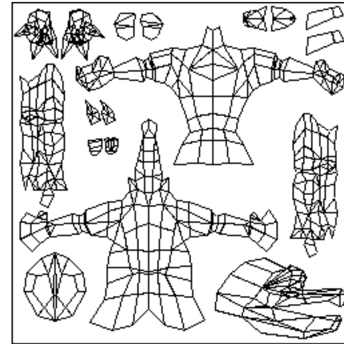


During rasterization interpolate the coordinate indices into the texture map

Case Studies: Low Poly Modeling

- With low polygon modeling, much of the detail is painted into the texture





Images courtesy of WildTangent, model and texture by David Johnson.

Texture Mapping Coordinates

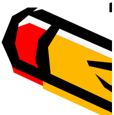
- Understanding U's and V's
- Managing the texture space
- Laying out and positioning of UV points

* poor continuity, confusing
* too many shapes and seams

a.

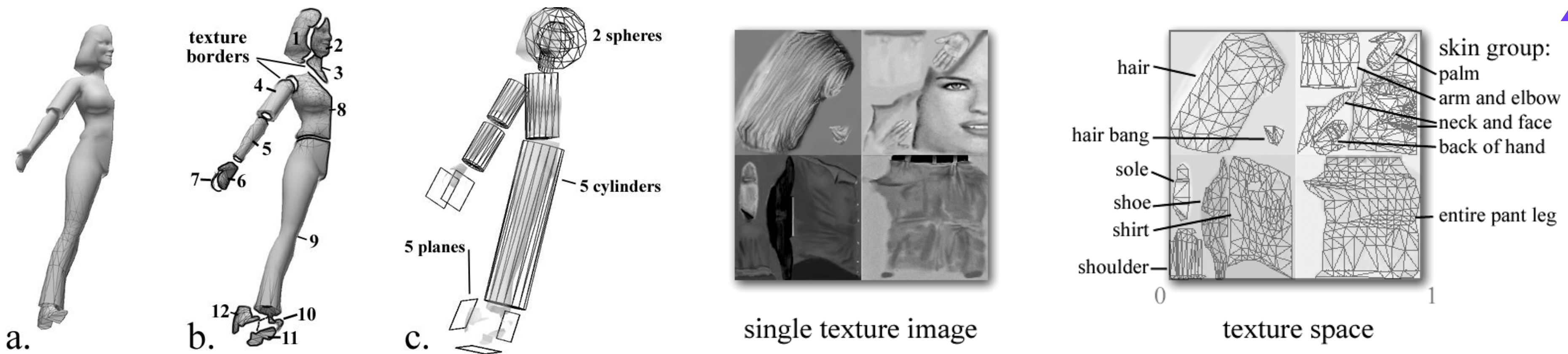
* easier to undersand and paint
* fewer seams

b.



Breaking Down Mesh Object for Mapping

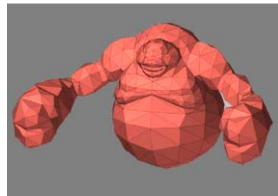
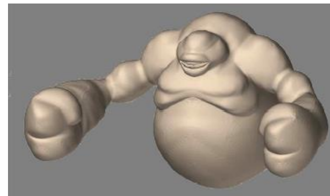
- Evaluate the 3D model for common areas

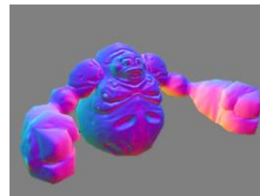


- Avoid duplication
  - Simplifies the work
  - Saving valuable texture space
  - Reduce the amount of texture borders

Applications beyond Texture Map Normal map: 3D Sculpting

- A low resolution model can be sculpted into a very detailed mesh.
- This can be used in game via normal maps



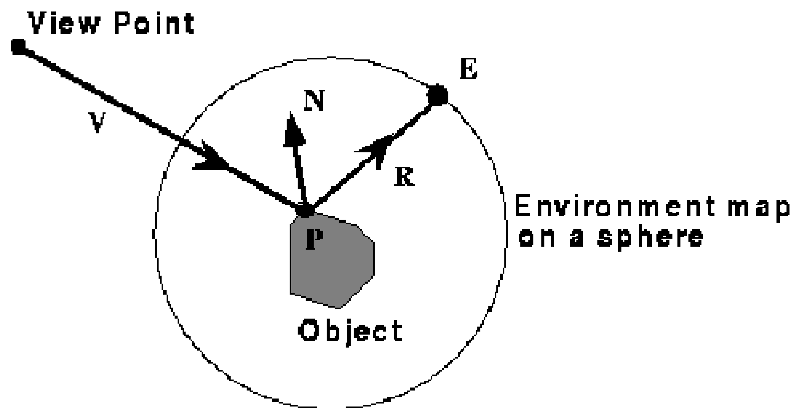
Images courtesy of Pixolgic.

Applications beyond Texture Map

Environment Maps

Use texture to represent reflected color

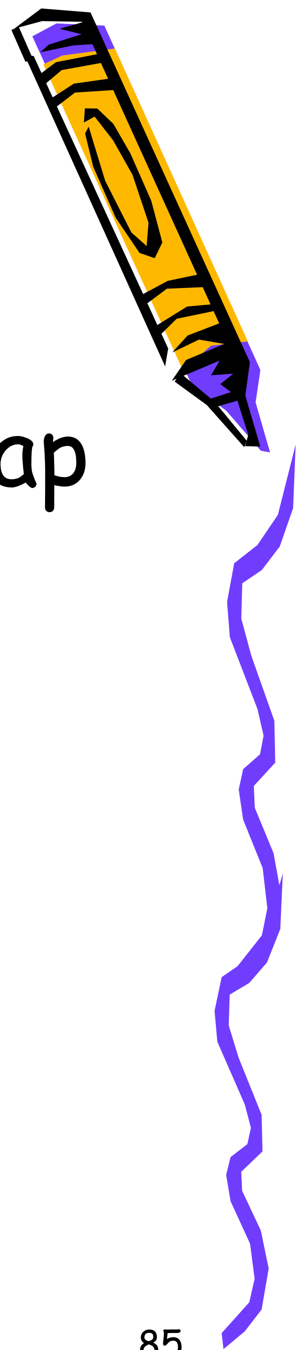
- Texture indexed by reflection vector
- Approximation works when objects are far away f … ct



Environment Maps

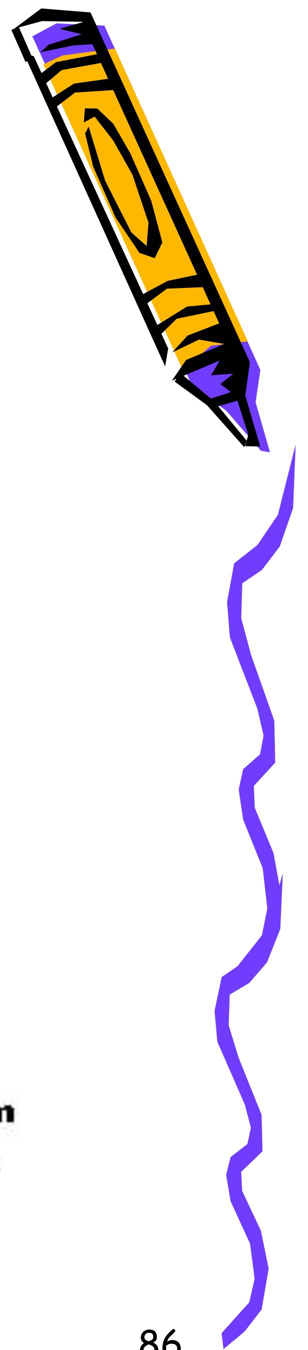
Using a spherical environment map

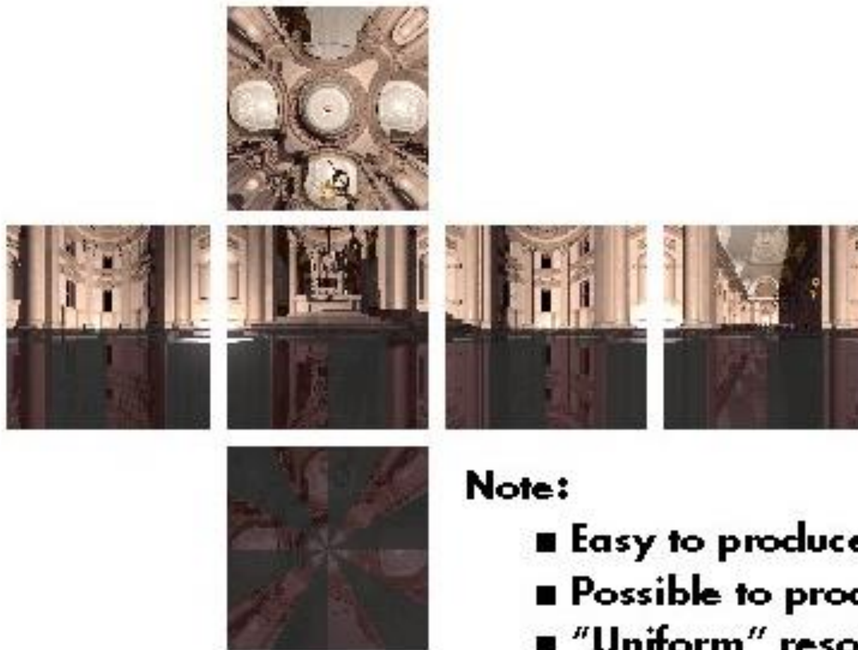
Spatially variant resolution



Environment Maps

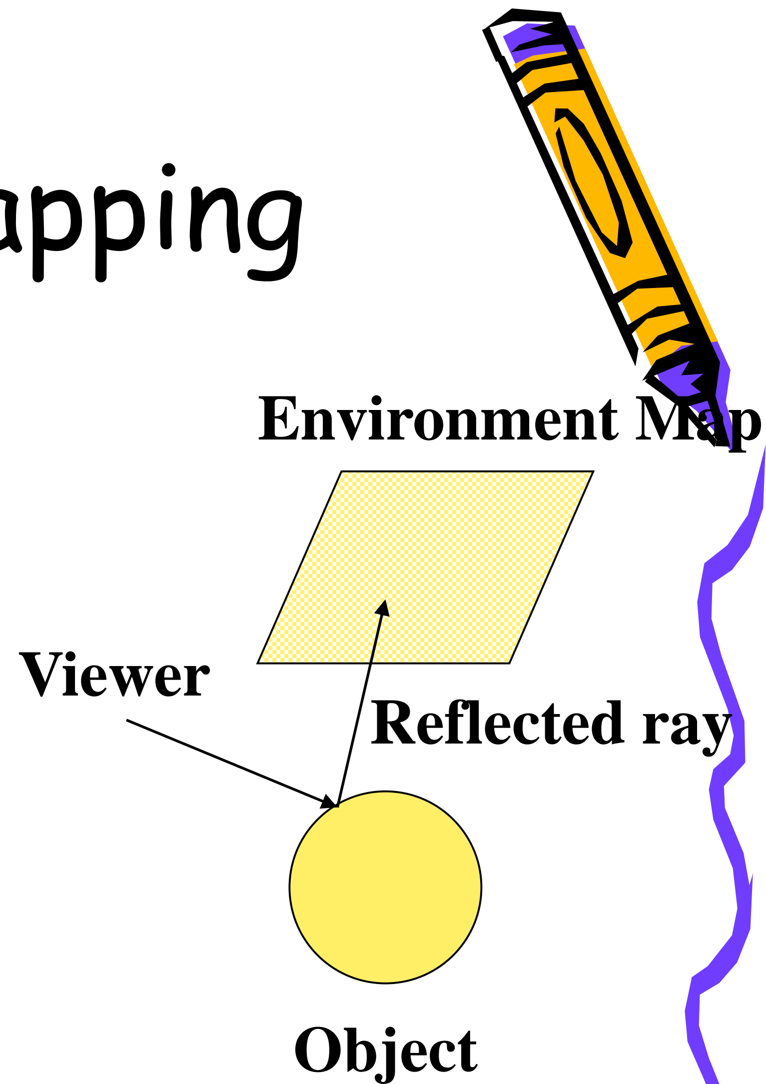
Using a cubical environment map



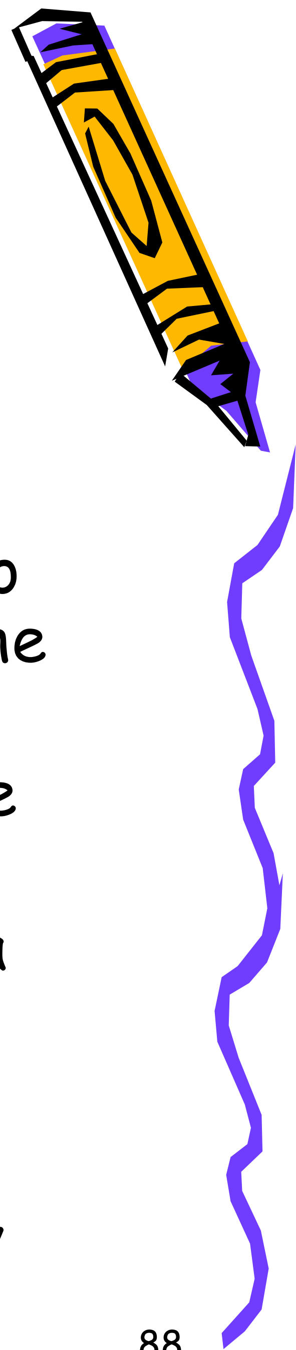


Environment Mapping

- Environment mapping produces reflections on shiny objects
- Texture is transferred in the direction of the reflected ray from the environment map onto the object
- Reflected ray: $\boldsymbol{R}=2(\boldsymbol{N}\cdot \boldsymbol{V})\boldsymbol{N}-\boldsymbol{V}$
- What is in the map?

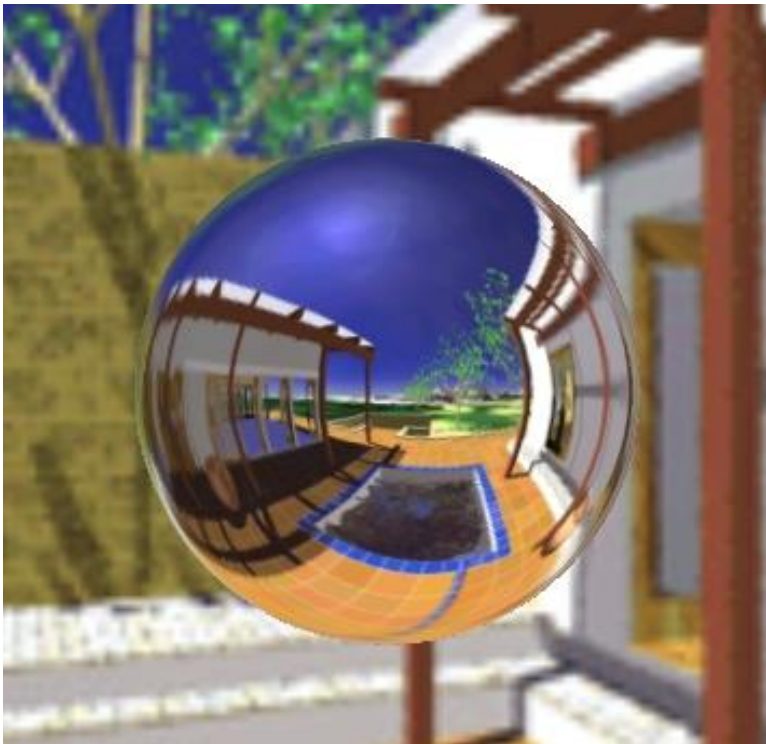


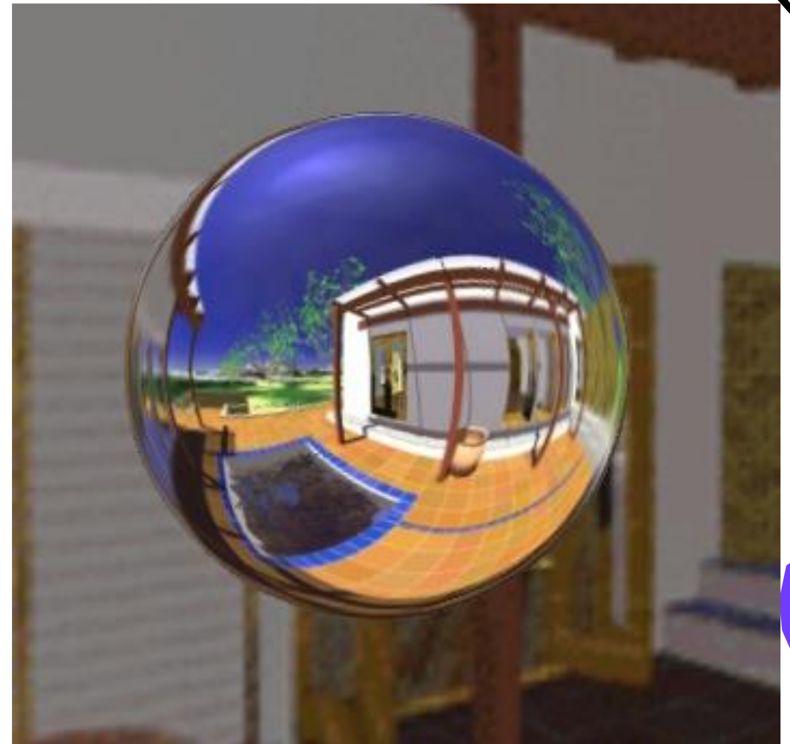
Approximations Made



- The map should contain a view of the world with the point of interest on the object as the eye
  - We can't store a separate map for each point, so one map is used with the eye at the center of the object
  - Introduces distortions in the reflection, but the eye doesn't notice
  - Distortions are minimized for a small object in a large room
- The object will not reflect itself
- The mapping can be computed at each pixel, or only at the vertices

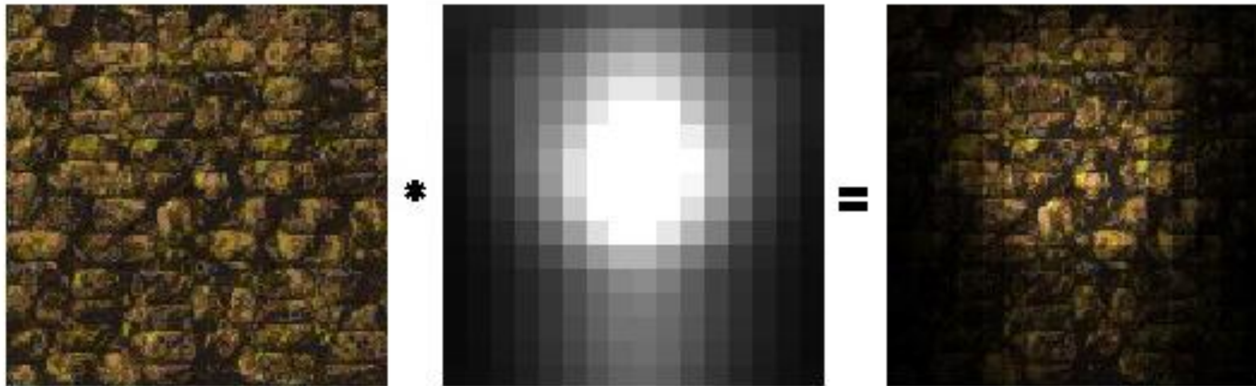
Example

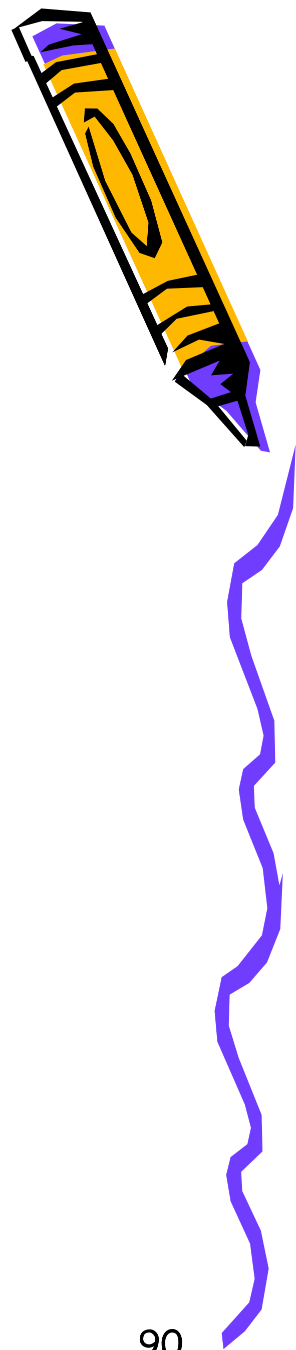




Illumination Maps

Use texture to represent illumination footprint

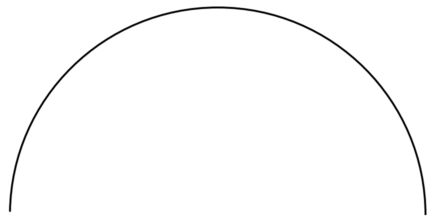




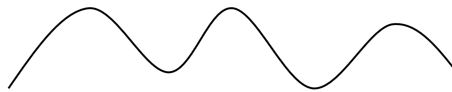
Bump Mapping

Use texture to perturb normals

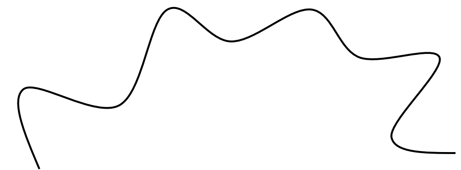
- creates a bump-like effect



+

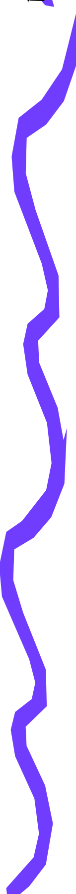


=



original surface **bump map** **modified surface**

$O(u,v)$ $B(u,v)$ $O'(u,v)$



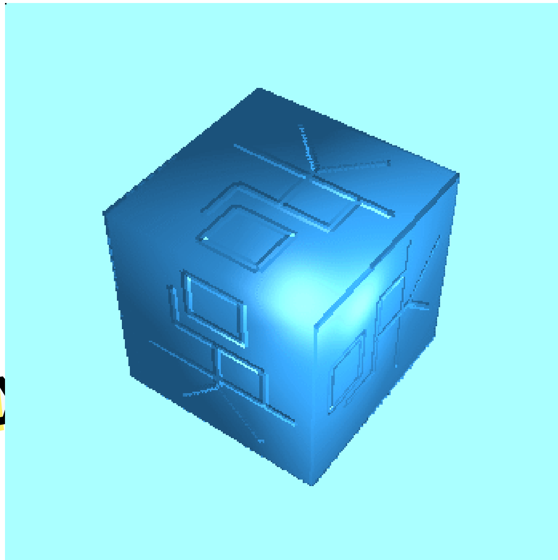
Does not change silhouette edges

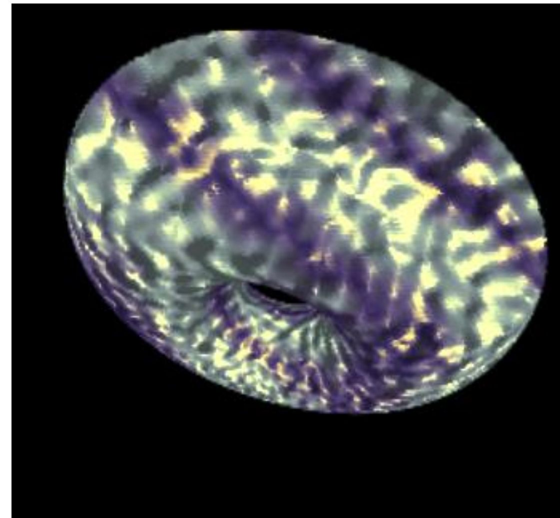
Bump Mapping

- Many textures are the result of small perturbations in the surface geometry
- Modeling these changes would result in an explosion in the number of geometric primitives.
- Bump mapping attempts to alter the lighting across a polygon to provide the illusion of texture.

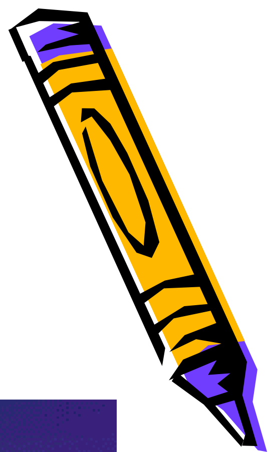
Bump Mapping

- This modifies the surface normals.

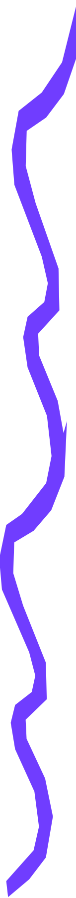




Bump Mapping

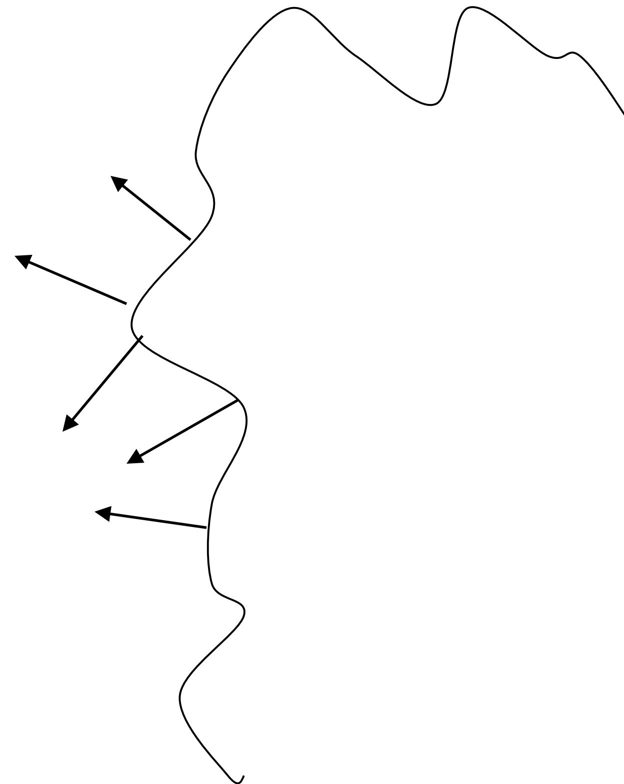


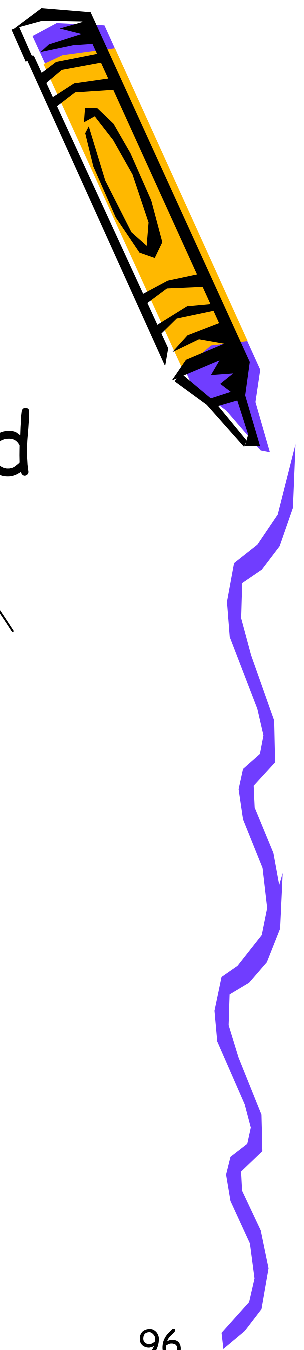
Bump Mapping

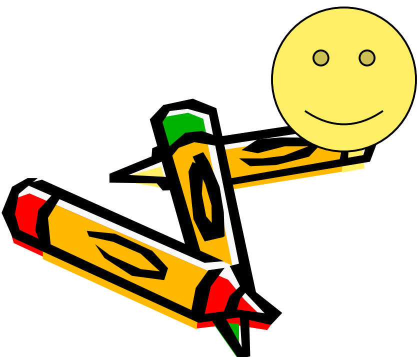


Bump Mapping

- Consider the lighting for a modeled surface.







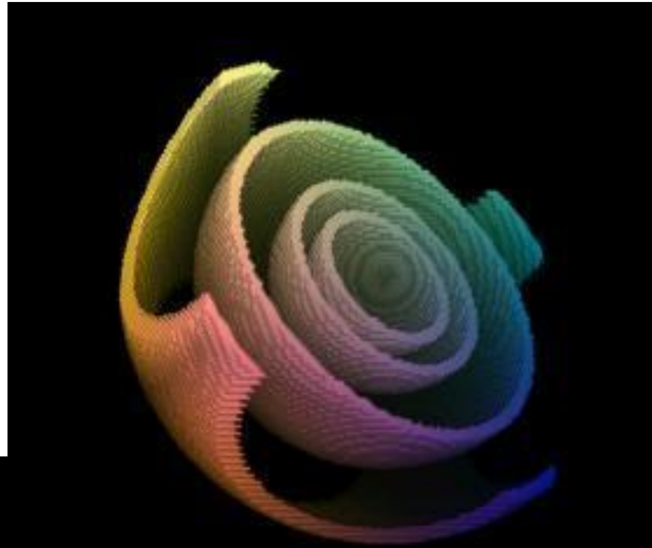
3D Textures

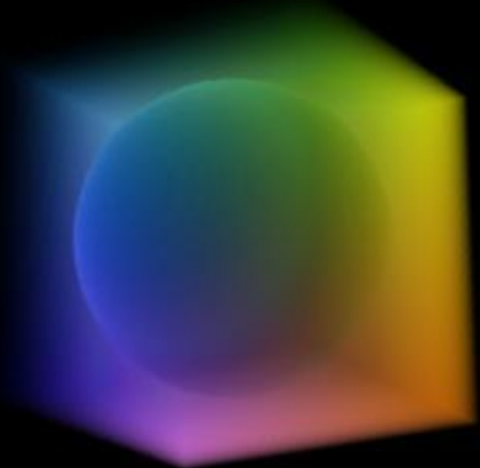
Use a 3D mapping
$$\left(x_o, y_o, z_o\right) \rightarrow \left(r, s, t\right)$$

Usually stored procedurall

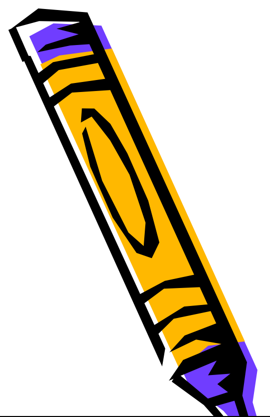
Can simulate an object carved from a material

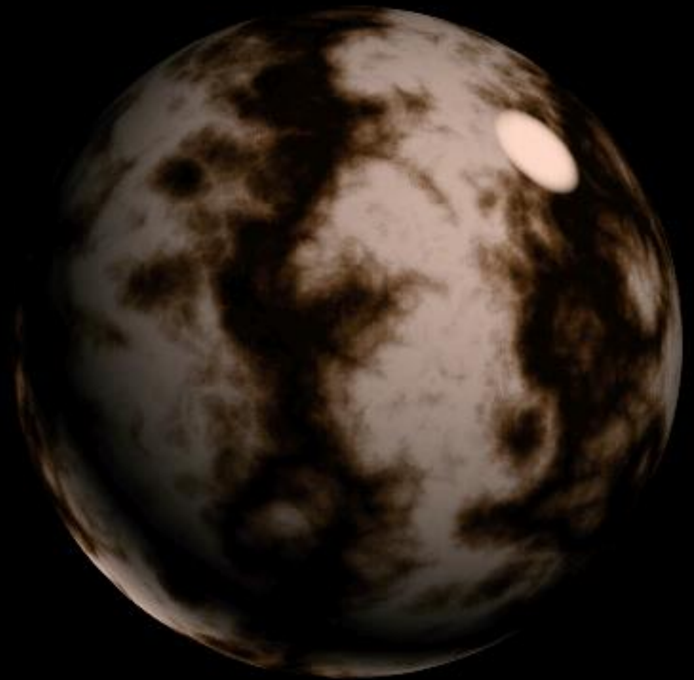
Examples of 3D texture





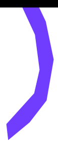
Turbulence





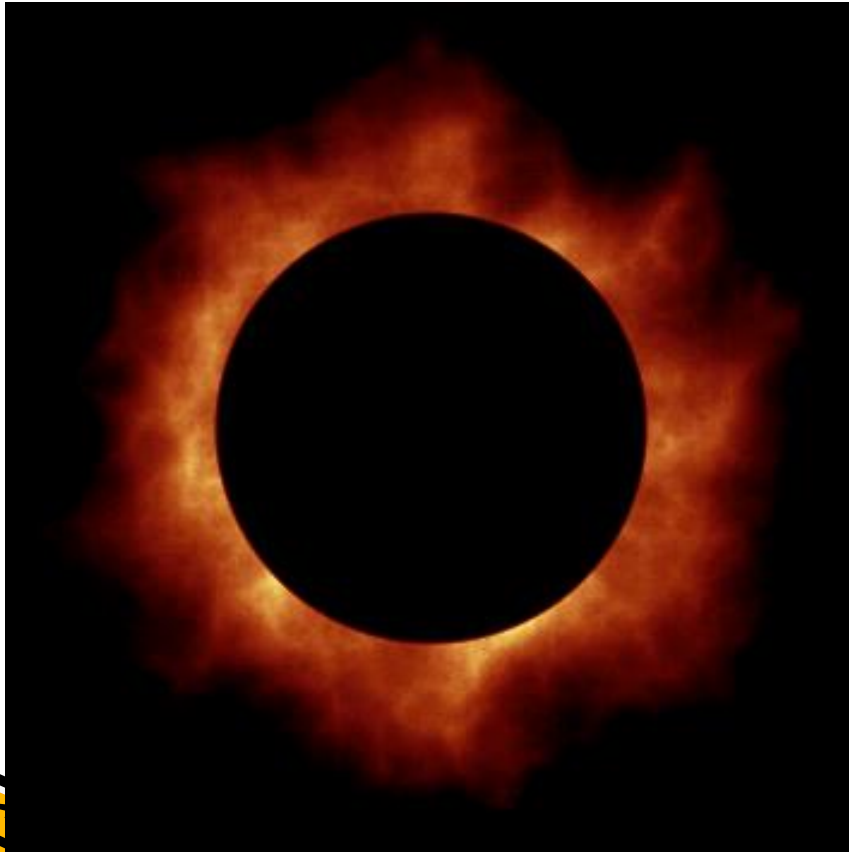
$turbulence(x)$

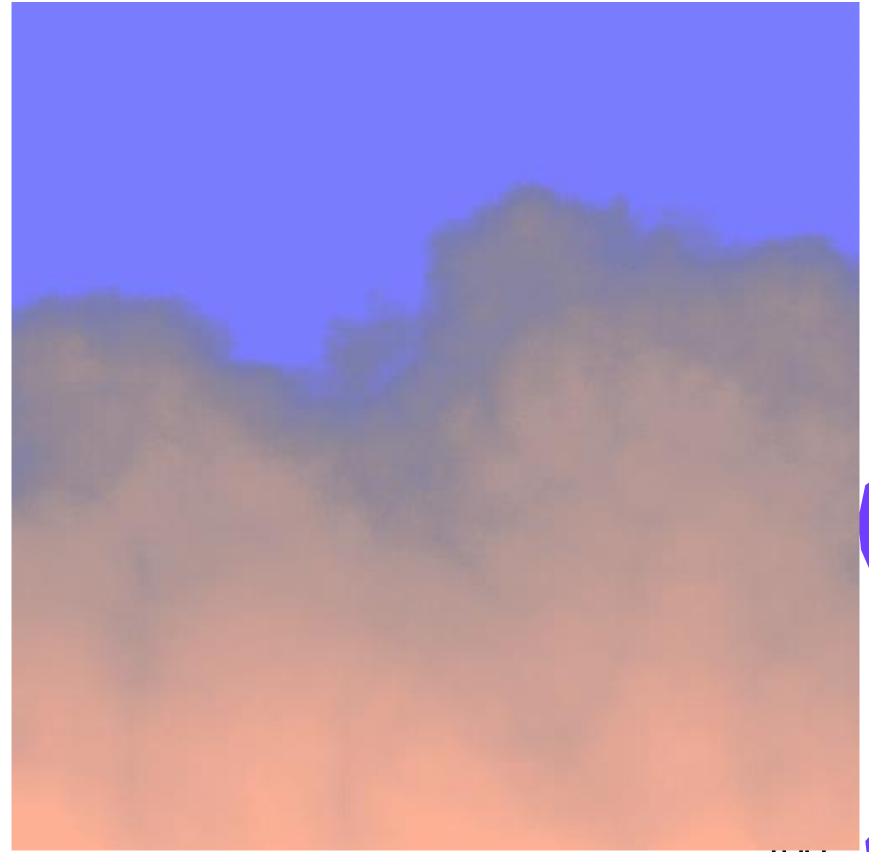
$\sin(x + turbulence(x))$

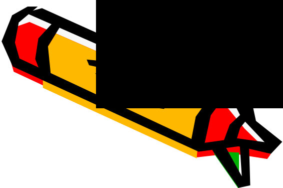


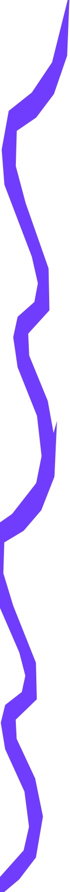
Animating Turbulence

Use an extra dimension as time

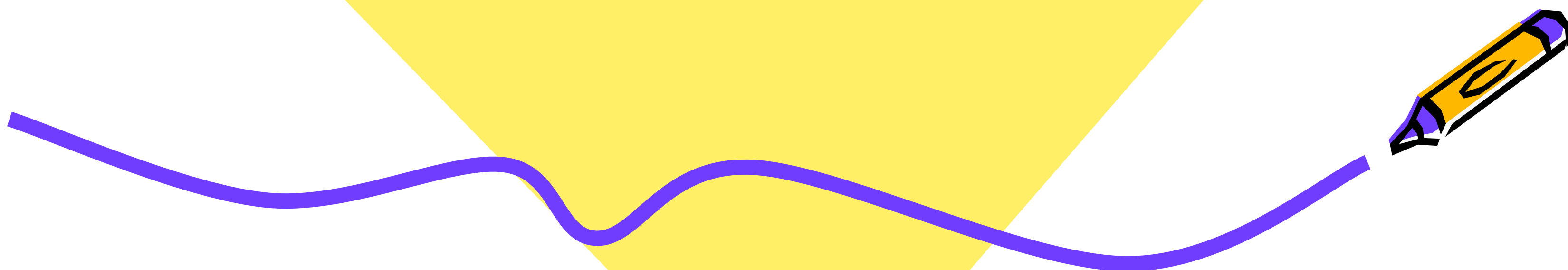






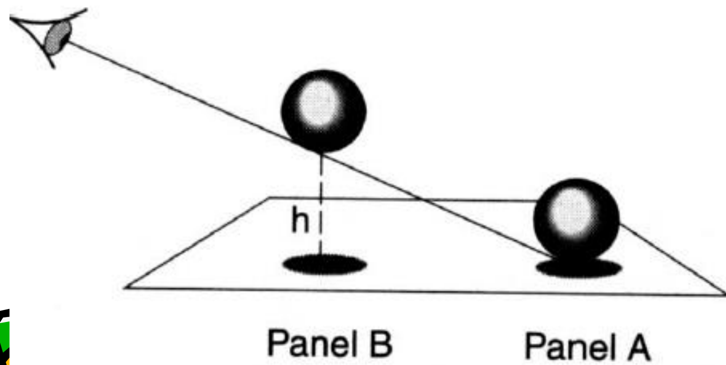


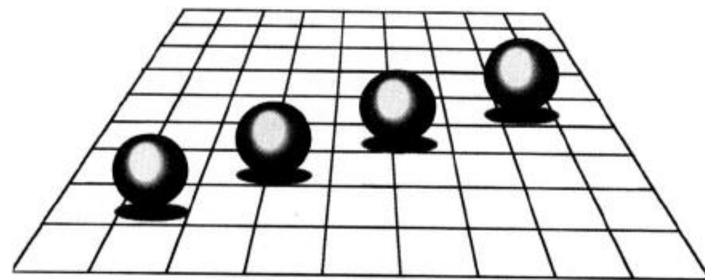
Shadow

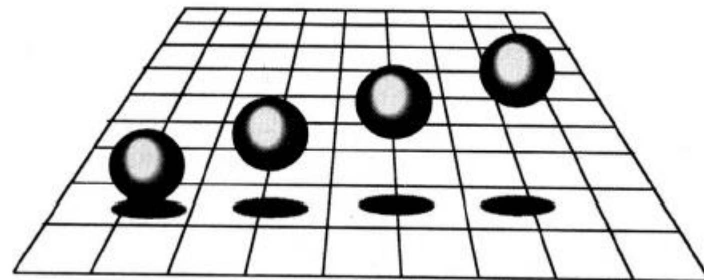


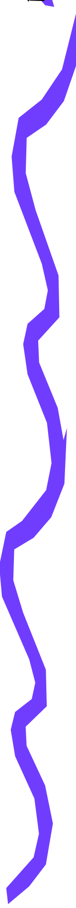
Course Note Credit: Some of slides are extracted from the course notes of prof. Mathieu Desburn (USC) and prof. Han-Wei Shen (Ohio State University).

Shadows as Depth cue



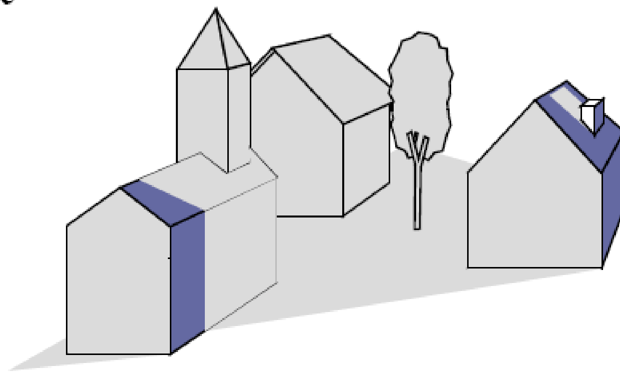
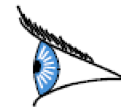
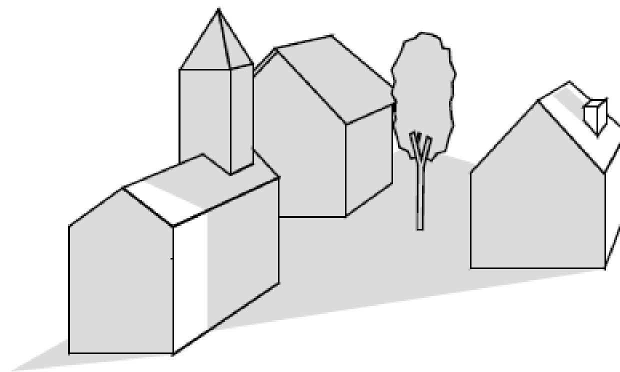
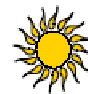






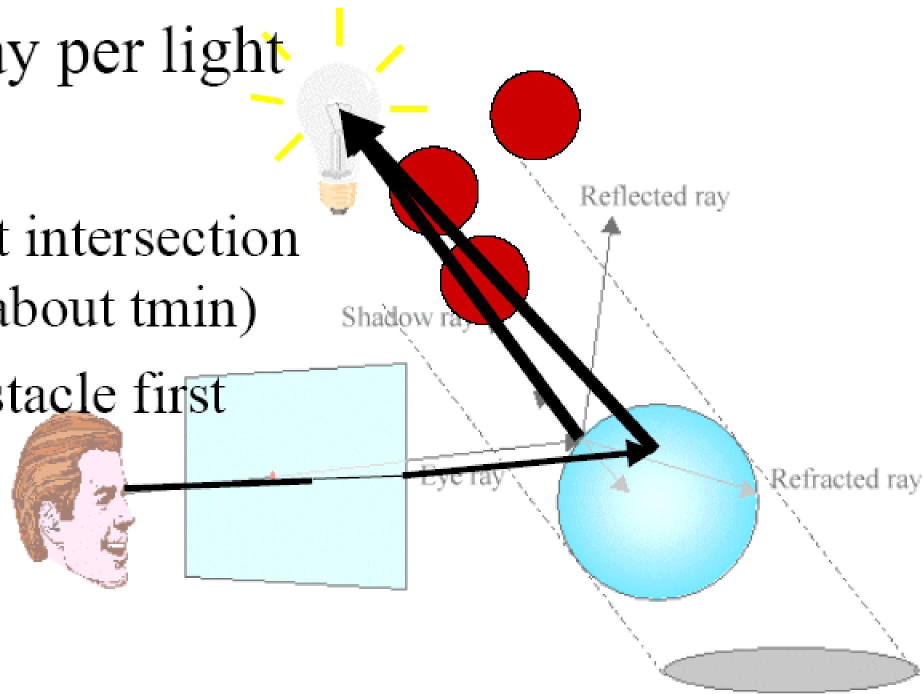
Duality of shadow & view

- A point is lit if it is visible from the light source
- Shadow computation very similar to view computation

Shadow Ray

- Ray from visible point to light source
- If blocked, discard light contribution
- One shadow ray per light
- Optimization?
  - Stop after first intersection (don't worry about tmin)
  - Test latest obstacle first



Shadow Maps

- Use texture mapping but using depth
- 2 passes (at least)
  - Compute shadow map from light sourc
    - Store depth buffer (shadow map)
  - Compute final image
    - Look up the shadow map to know if points are in shadow

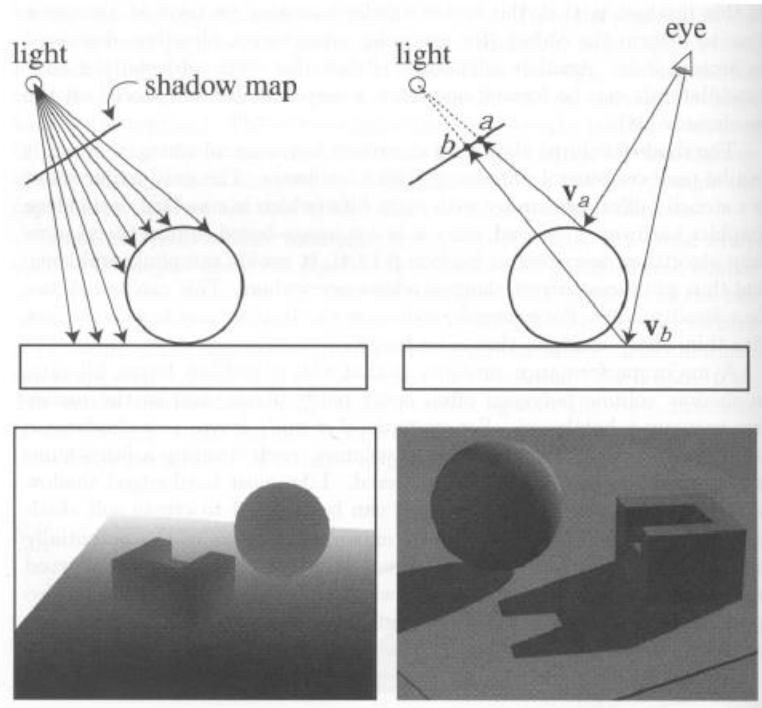
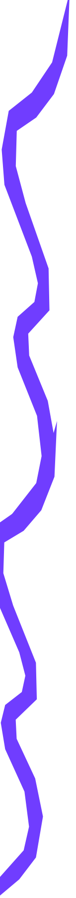


Figure from Foley et al. "Computer Graphics Principles and Practice"

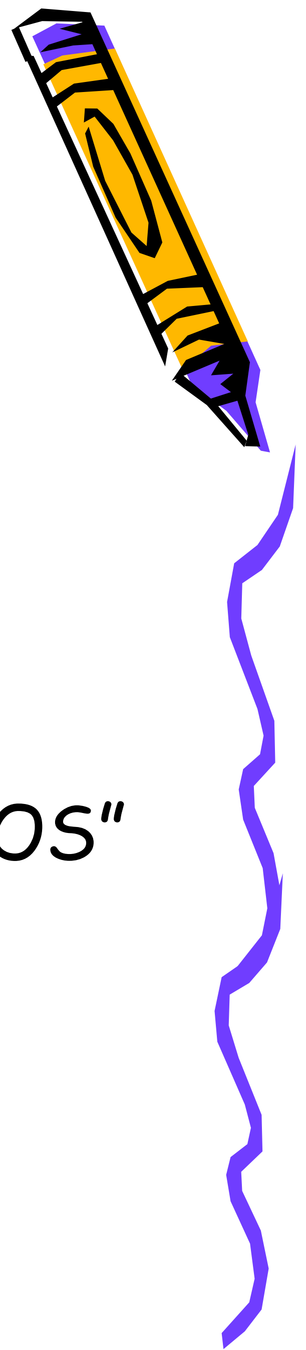
What is Game Engine

- Game Engine vs. real engine of a car
  - Non game specific technology
- Game vs. Car
  - all the content (models, animations, sounds, AI, and physics) which are called 'assets'
  - the code required specifically to make that game work, like the AI, or how the controls work

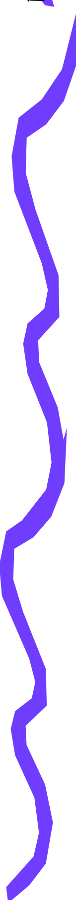


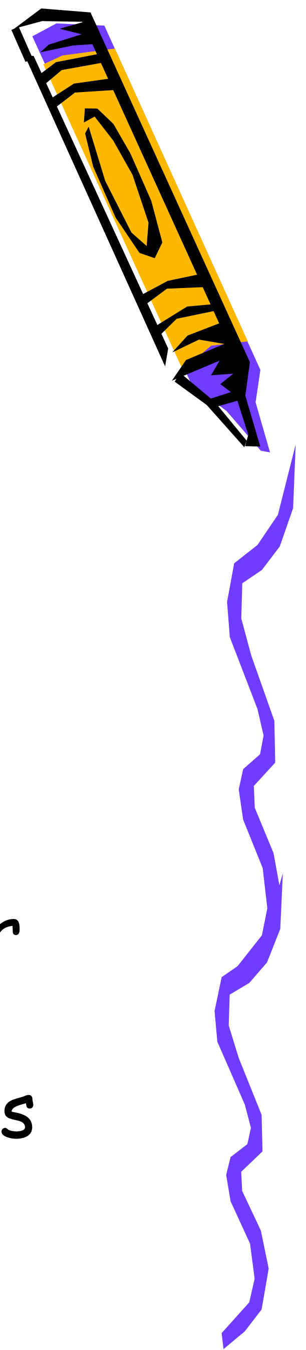
Game Engine

- Reusable software components within different games
- enable simplified, rapid development of games in a data-driven manner
- sometimes called "game middleware or OS"



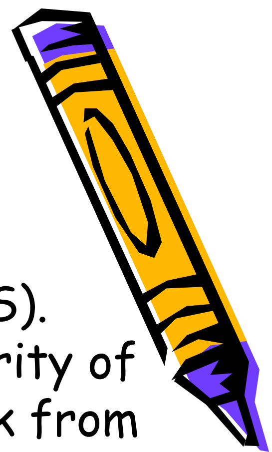
- software system designed for the creation and development of video games. There are many game engines that are designed to work on video game consoles and desktop operating systems such as Microsoft Windows, Linux, and Mac OS X. The core functionality typically provided by a game engine includes a rendering engine ("renderer") for 2D or 3D graphics, a physics engine or collision detection (and collision response), sound, scripting, animation, artificial intelligence, networking, streaming, memory management, threading, and a scene graph. The process of game development is frequently economized by in large part reusing the same game engine to create different games.



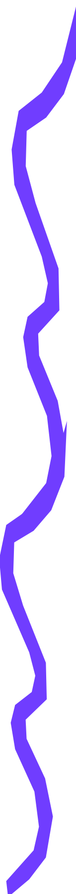


- Modern game engines are some of the most complex applications written, frequently featuring dozens of finely tuned systems interacting to ensure a precisely controlled user experience. The continued evolution of game engines has created a strong separation between rendering, scripting, artwork, and level design. It is now common (as of 2003), for example, for a typical game development team to have several times as many artists as actual programmers

Game Engine History



- The term "game engine" arose in the mid-1990s, especially in connection with 3D games such as first-person shooters (FPS). (*See also:* first-person shooter engine). Such was the popularity of id Software's *Doom* and *Quake* games that, rather than work from scratch, other developers licensed the core portions of the software and designed their own graphics, characters, weapons and levels—the "game content" or "game assets." Separation of game-specific rules and data from basic concepts like collision detection and game entity meant that teams could grow and specialize.
- Later games, such as *Quake III Arena* and Epic Games's 1998 *Unreal* were designed with this approach in mind, with the engine and content developed separately. The practice of licensing such technology has proved to be a useful auxiliary revenue stream for some game developers, as a single license for a high-end commercial game engine can range from US$10,000 to millions of dollars, and the number of licensees can reach several dozen companies (as seen with the Unreal Engine). At the very least, reusable engines make developing game sequels faster and easier, which is a valuable advantage in the competitive video game industry.

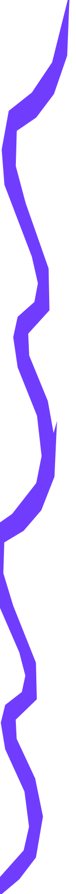


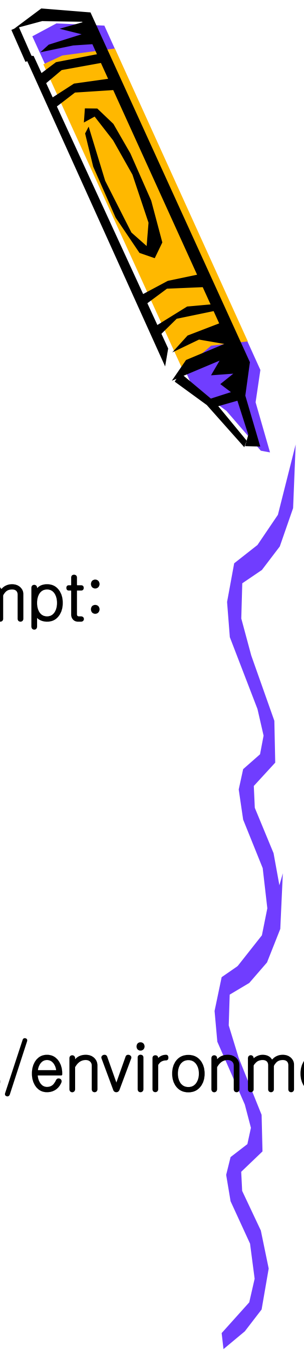
Graphics Engine

- The Renderer (Graphics Engine)
  - RealmForge, Ogre, Power Render, Crystal Space, Genesis3D, and JMonkey Engine
  - scene graph, which is an object-oriented representation of the 3D game world

Panda3D

- http://panda3d.org/
- a library of subroutines for 3D rendering and game development.
- Game development with Panda3D usually consists of writing a Python program that controls the Panda3D library.
- emphasis is on supporting a *short learning curve* and *rapid development*.





To start Panda3D, create a text file and save it with the .py extension. PYPE (available at http://sourceforge.net/projects/pype/), SPE and IDLE are Python-specific text-editors, but any text editor will work. Enter the following text into your Python file:

```
import direct.directbase.DirectStart
run()
```

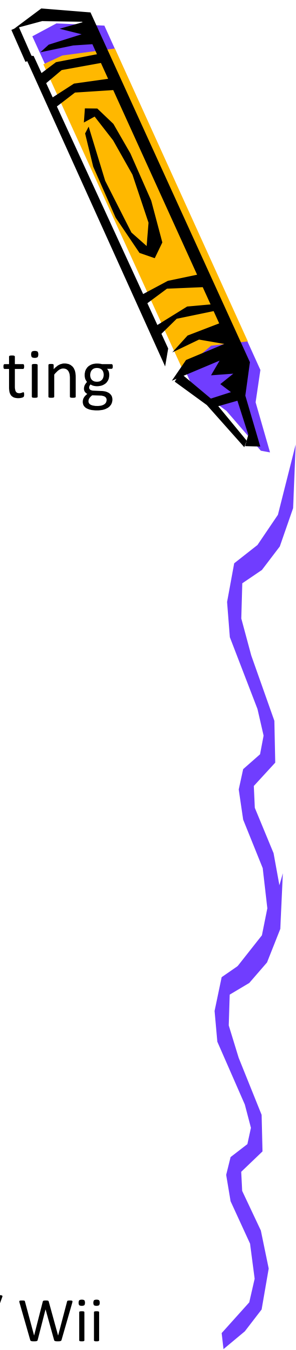
To run your program, type this at the command prompt:

ppython filename.py

```
import direct.directbase.DirectStart

#Load the first environment model
environ = loader.loadModel("models/environme
environ.reparentTo(render)
environ.setScale(0.25,0.25,0.25)
environ.setPos(-8,42,0)
#Run the tutorial
run()
```

Test1.py



What is Unity?

- Unity is a multi-platform, integrated IDE for scripting games, and working with 3D virtual worlds
- Including:
  - Game engine
    - 3D objects / lighting / physics / animation / scripting
  - Accompanying script editor
    - MonoDevelop (win/mac) << RECOMMENDED TO USE
    - Unitron (Mac) / UniSciTE (Windows) << DEFAULT
    - Can also use Visual Studio (Windows)
  - 3D terrain editor
  - 3D object animation manager
  - GUI system
  - Executable exporter many platforms: native application / web player / iPhone / Android / Wii

Unity – main interface components

Pivot Local

Layers Layout

Scene

Textured RGB

1

Hierarchy

FPS
AllSeagulls
Daylight Water
Directional Back light
Directional Sun light
First Person Controller Prefab
HeronColliders
HeronPrefab
HeronPrefab
LevelObjects
Performance
SeaFloor
Terrain
UnderWater Water Surface
Waves

2

Inspector

First Person Controller Prefab

Tag Player Layer Default

Discon Select Reconnect Apply

Transform

Position X 1043.066 Y 51.10947 Z 692.8281

Rotation X 0 Y 346.453 Z 0

Scale X 1.69211 Y 1.69211 Z 1.69211

FPSWalker (Script)

Script	FPSWalker
Speed	15
Jump Speed	8
Gravity	20

Character Controller

Height	2
Radius	0.4
Slope Limit	45
Step Offset	0.4
Skin Width	0.05
Min Move Distance	0

Center

Mouse Look (Script)

Script	MouseLook
Axes	MouseX
Sensitivity X	15
Sensitivity Y	0
Minimum X	-360
Maximum X	360
Minimum Y	0
Maximum Y	0

3

Game

Free Aspect Maximize on Play Gizmos Stats

4

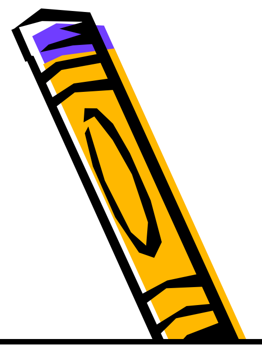
Project

Create

AirplaneRuins
Birds
Bridges
Bunkers
Editor
First Person Controller Prefab
Fish
Islands
LightmapPSD
LightmapWithFog
LongBoat_NC_Untied
New Terrain
Pro Standard Assets
Scripts
seaFoamCoast
Sounds
Standard Assets
Terrain Demo Assets
Water

5

Unity – main interface components

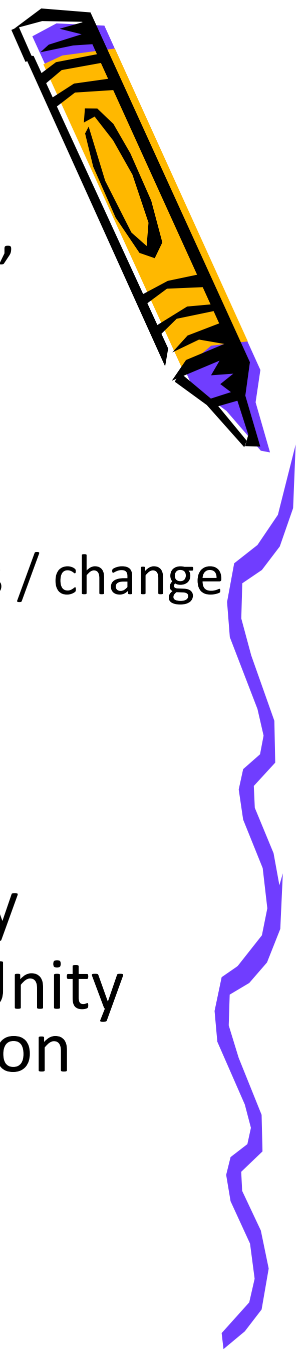


- 1 – Scene   ***Scene = Hierarchy = same, just diff. views***
  - Editable (design-time) 3D game objects in the current scene
- 2 – Hierarchy
  - Text list of game objects and sub-objects in the current scene
- 3 – Inspector
  - Properties for currently selected Game Object
- 4 – Game
  - Preview how game will look when exectuting
- 5 – Project
  - Contents of Project 'assets' folder (i.e. files in that folder)
  - library of scripts, digital media files, and scenes

Scripting

- Unity implements a MONO compiler
- Scripts can be written in
  - JavaScript
    - Note – most introductory tutorials are written in Javascript – for those learning programming its fine
  - C#
    - Very similar to Java, Unity can be integrated with the Microsoft Visual Studio editor, to get full benefits of code completion, source version control etc.
    - Serious developers work in C# …
  - Also BOO (like Python) – little development is this …

Scenes

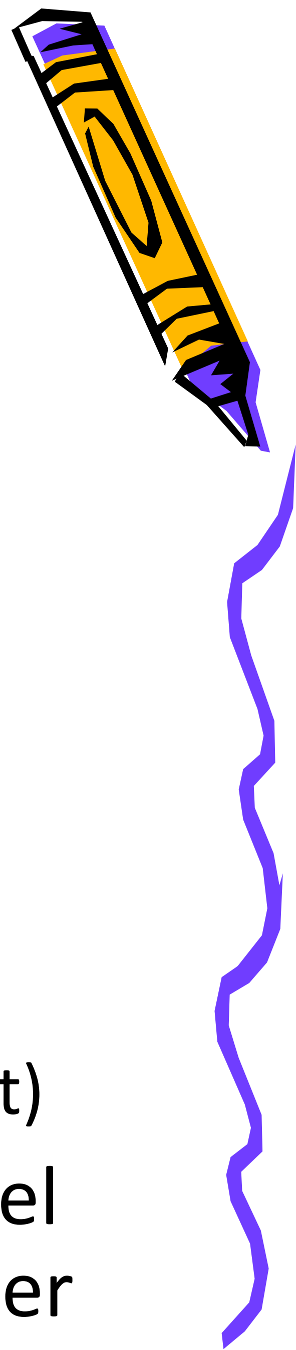


- A unity "scene" is essentially a "level" or "screen"

- Typical game
  - Welcome / main menu screen
    - Buttons: play game / see high scores / read instructions / change input settings
  - Level 1 / Level complete / Level 2 etc…
  - Game Over / Enter details for new High Score …

- All the above would be separate "scenes" in unity
- Some scenes may be entirely based around the Unity GUI scripts / components – i.e. be text / buttons on screen

Project Assets



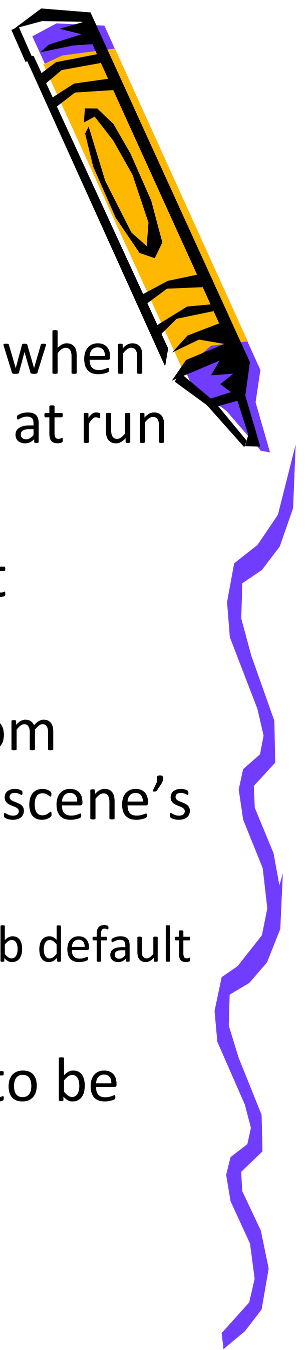
- The Assets folder for each Unity project contains:
  - Scenes
  - Media assets
    (images, sounds files, 3D models)
  - Script files
  - “packages”
    (collections of unity assets, ready to import)
- The contents of the Unity “Project” panel reflect the contents of the “Assets” folder

Game Objects – in current ‘scene’

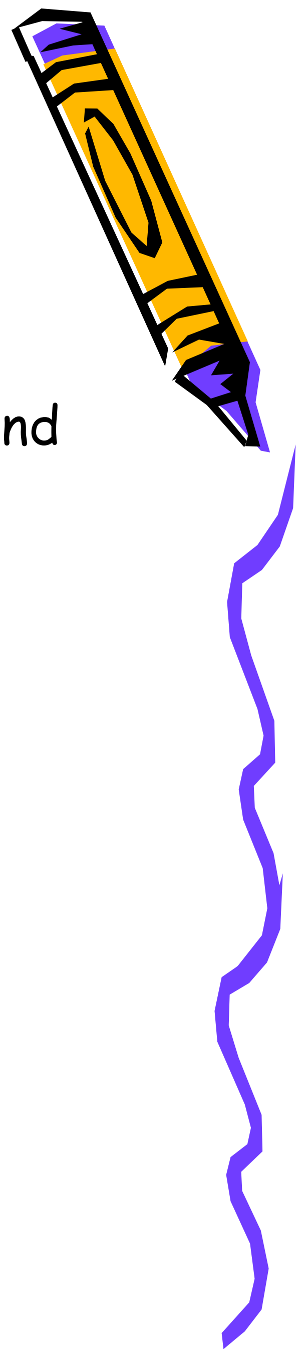
- Everthing in a scene is either a Game Object
  - or a component INSIDE a Game Object
- Every Game Object has at least 1 COMPONENT
  - Its TRANSFORM – an object’s position, scale, rotation
  - Other components depend on object type (audio, mesh, material, script etc.)

- Game objects can be in a HIERARHCY – so an object can be a sub-object of another object
  - E.g. an “arm” object can be a sub-object of a “body” object etc.

Unity “Prefabs” powerful concept …



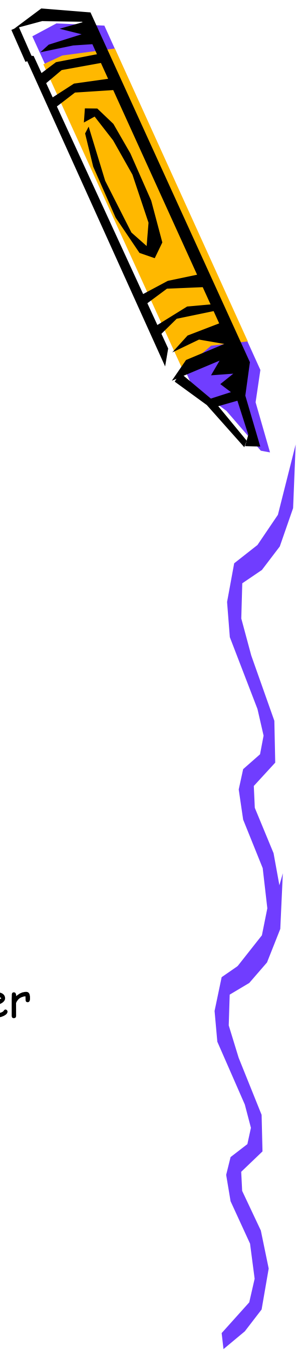
- Since object-oriented (although this is partially hidden when scripting in JavaScript) instances can be INSTANTIATED at run time
- Unity uses the term PREFAB for a pre-fabricated object template (i.e. a class combining 3D objects and scripts)
- At DESIGN TIME (in editor) a prefab can be dragged from Project window into the Scene window and added the scene’s hierarchy of game objects
  - The object can then be edited (i.e. customised from the prefab default settings) if desired
- At RUN TIME a script can cause a new object instance to be created (instantiated) at a given location / with a given transform set of properties



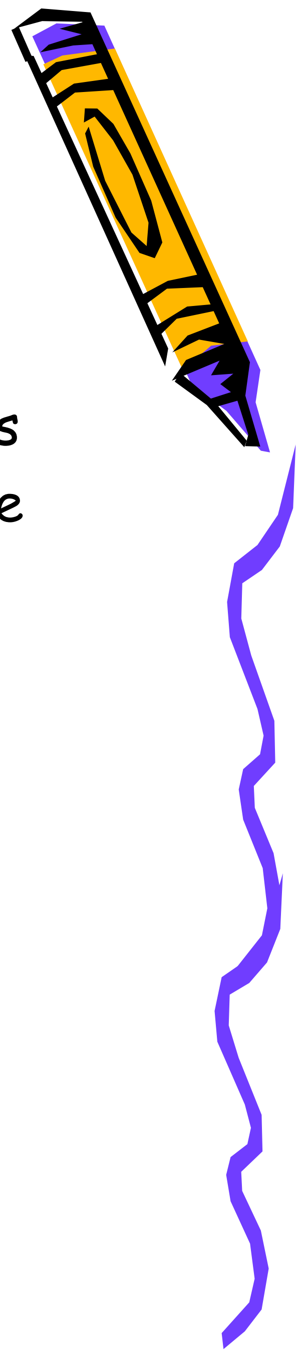
Unity 3D Terrain Editor

- Create terrain by selecting brush type, brush size and opacity and then sculpting topology
- Set maximum height and smooth corners
- Textures loaded to paint texture onto terrain
- First texture acts as background to subsequent
- Paint on trees and other smaller items e.g grass.

Unity 3D Lights and Cameras



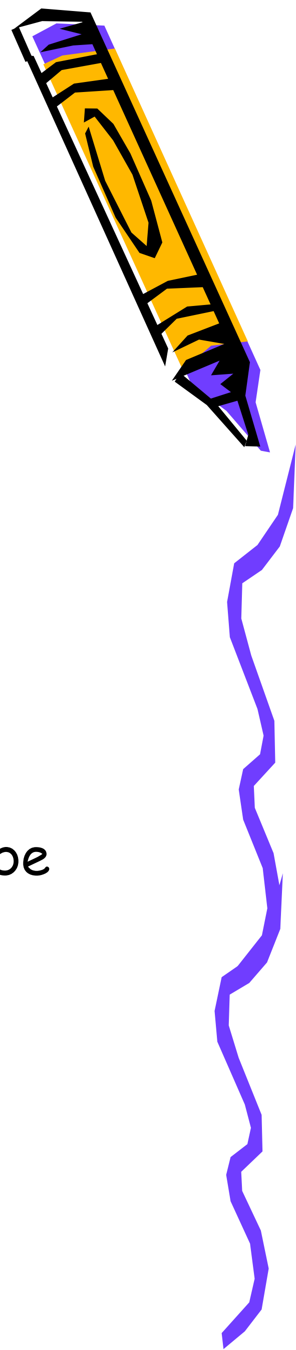
- Lights
  - Directional
  - Point
  - Spot
  - Lights can be parented to other game objects
- Cameras
  - One default camera
  - First Person Controller includes camera
  - Camera acts as an Audio Listener in the scene
  - Remove default camera to only have one Audio Listener
  - Cameras can be parented to other game objects



Unity 3D Textures

- Materials form the basic starting point for textures
- Textures should be in the following format to enable 'tiling'.
  - Square and the power of two
  - 128 x 128, 256 x 256, 512 x 512, 1024 x 1024
- Shaders control the rendering characteristics of textured surface

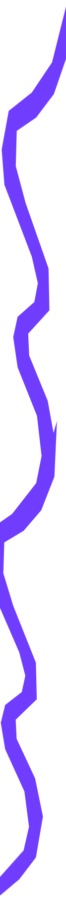
Physics and Collision Detection

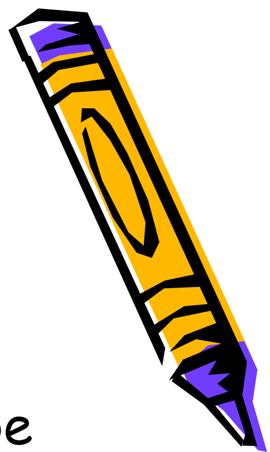


- Physics component
  - Mass
  - Gravity
  - Velelocity
  - Friction
- Physics component added to game object.
- Collision detection provided for most objects - can be customized with sphere colliders and mesh colliders
- Mesh colliders most computationally expensive
- Also level of detail LOD is handled by game engine

Scripting

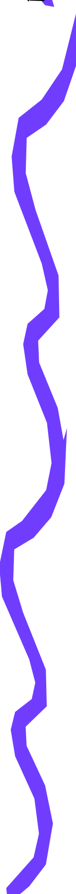
- http://unity3d.com/learn/tutorials/topics/scripting





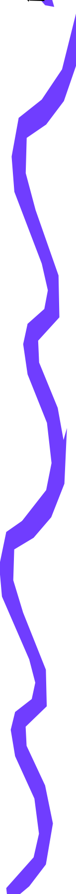
Sky Boxes and Effects

- Skybox - cubemap - six textures placed inside a cube
- Rendered seamlessly to appear as surrounding sky and horizon
- Not an object position a player can visit
- Only visible in the Game View panel
- Water effects created by an animated material applied to a surface



Audio Effects

- Audio requires an Audio Source and an Audio Listener in the scene
- Stereo sound treated as as ambient constant volume and continuously playing in the scene (looped enabled)
- Mono sound treated as spatial - gets louder or softer depending on player's position relative to the audio source position
- Supported formats .wav, .mp3, .aiff, .ogg



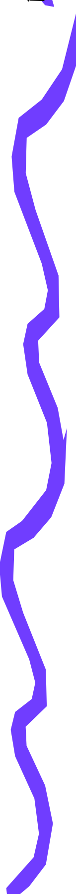
Unity 3D Terrain Editor

Set Heightmap resolution

Please note that modifying the resolution will clear the heightmap, detail map or splatmap.

Setting	Value
Terrain Width	1000
Terrain Height	600
Terrain Length	1000
Heightmap Resolution	513
Detail Resolution	1024
Control Texture Resolution	512
Base Texture Resolution	1024

Import



Inspector

Terrain

Tag Untagged Layer Default

Transform

Position

X	0	Y	0	Z	0

Rotation

X	0	Y	0	Z	0

Scale

X	1	Y	1	Z	1

Terrain (Script)

Raise Height

Hold down shift to lower height.

Brushes

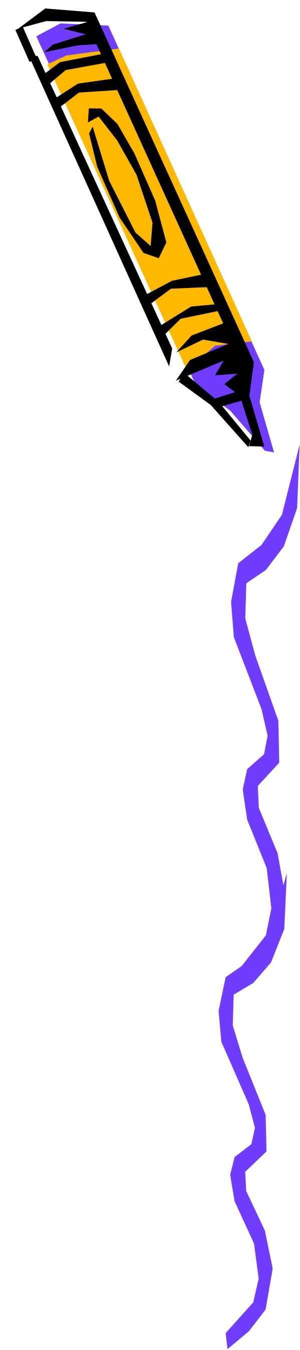
Settings

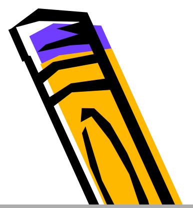
Brush Size 25

Opacity 0.5

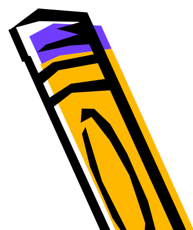
Terrain Collider

Material	None (Physic Material)
Is Trigger	☐
Terrain Data	New Terrain
Create Tree Colliders	☑





Scene
Textured
RGB
y
z
x
Persp



Scene
Textured
RGB
Persp