Mike Bailey
Oregon State University

mjb@cs.oregonstate.edu

vancouver
SIGGRAPH2014
The 41st International Conference and Exhibition
on Computer Graphics and Interactive Techniques

Seminar Goals

- Provide a background for everything else you will see at SIGGRAPH 2014
- Create a common understanding of computer graphics vocabulary
- Help appreciate the images you will see
- Get more from the Exhibition
- Provide pointers for further study

Mike Bailey

- Professor of Computer Science, Oregon State University

- Has worked at Sandia Labs, Purdue University, Megatek,
San Diego Supercomputer Center (UC San Diego), and OSU

- Has taught over 5,500 students in his classes

- mjb@cs.oregonstate.edu

Specific Topics

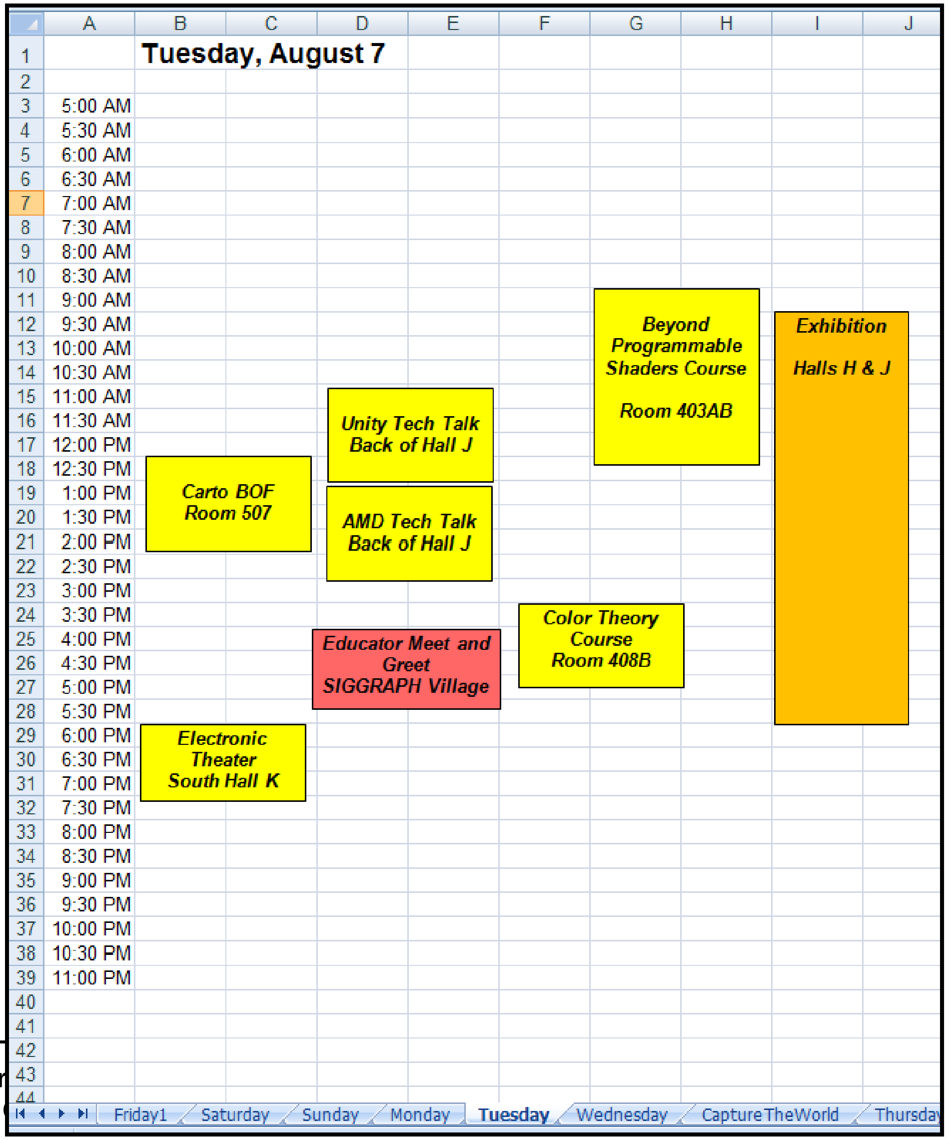
- The Graphics Process
- How to Attend SIGGRAPH
- Graphics Hardware
- Modeling
- Rendering
- Animation
- Finding More Information


How to Attend
SIGGRAPH

You can't see it all, so …

Think Strategically -- Make a Plan, Make a Schedule, Set Priorities !

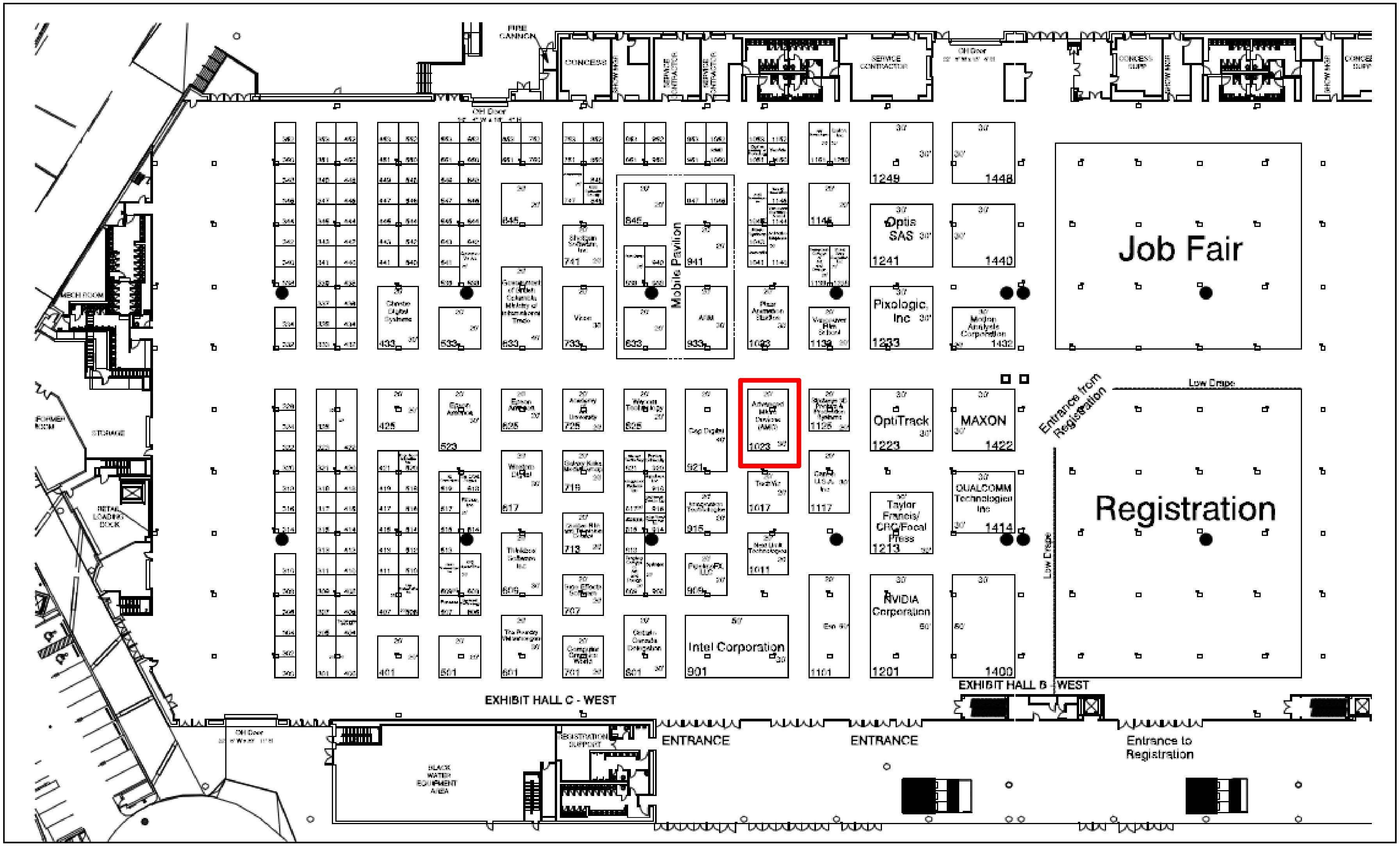
Your time is valuable.



In general, rank your top 3 things you want to see for each timeslot. Then, if one session is not as useful as you'd thought it would be, quickly move to your next priority.

Remember to give priority points to the things you can't "re-live" after it has happened !

OMG – Where do I Start in the Exhibition?



Exhibition Strategy

- Look at the list of vendors in the Conference Locator
- Make a list of the ones you *really* must see and sort the list by booth number
- Booth numbers are XXYY, where XX is the Aisle # and

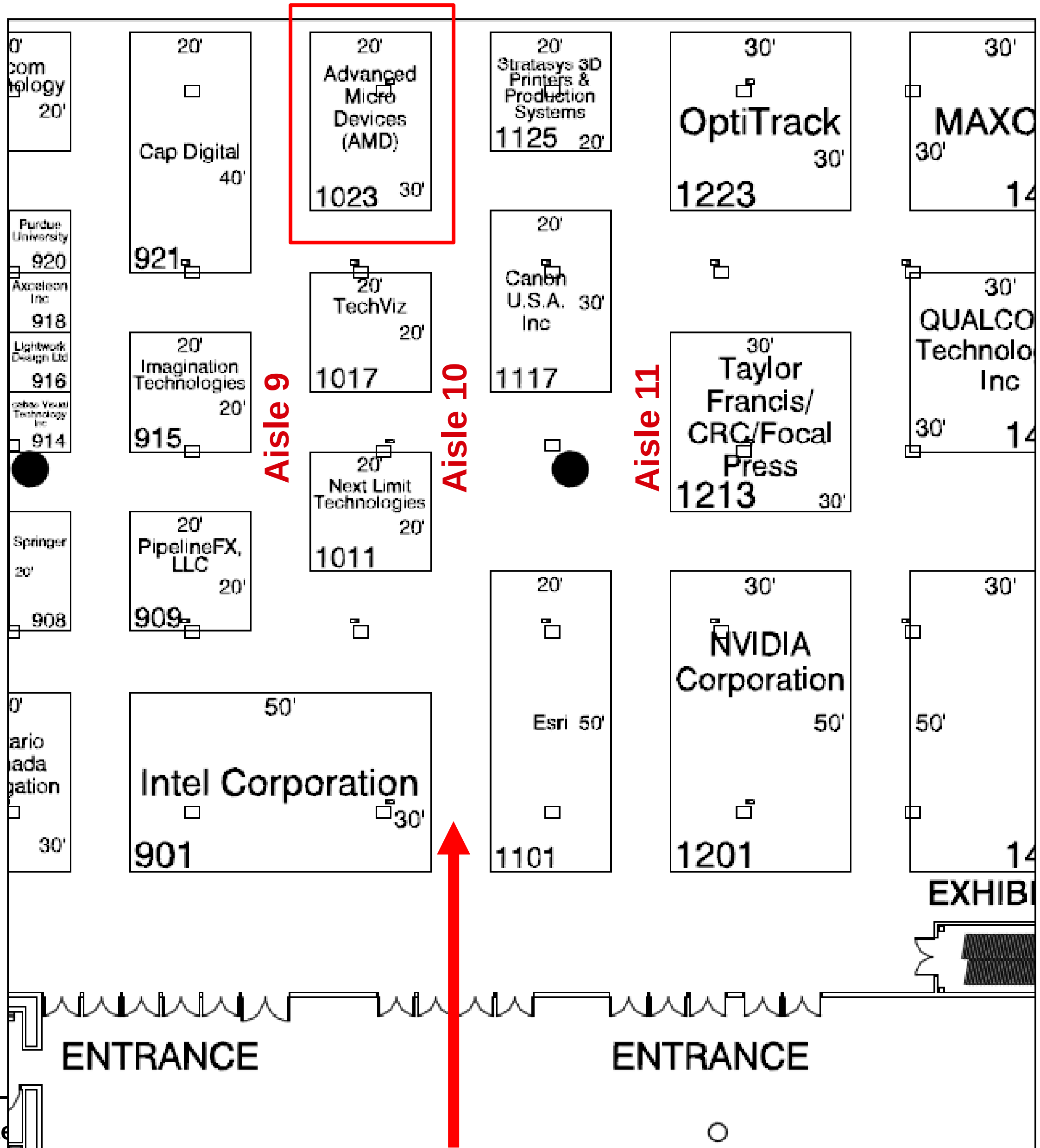
  YY is ($^1/_5$)*the number of feet from the front
- For example, AMD = booth 1023, which is Aisle 10; 5*23 = 115 feet from the front
- Start at one end of the floor and work your way across

Sorted by name

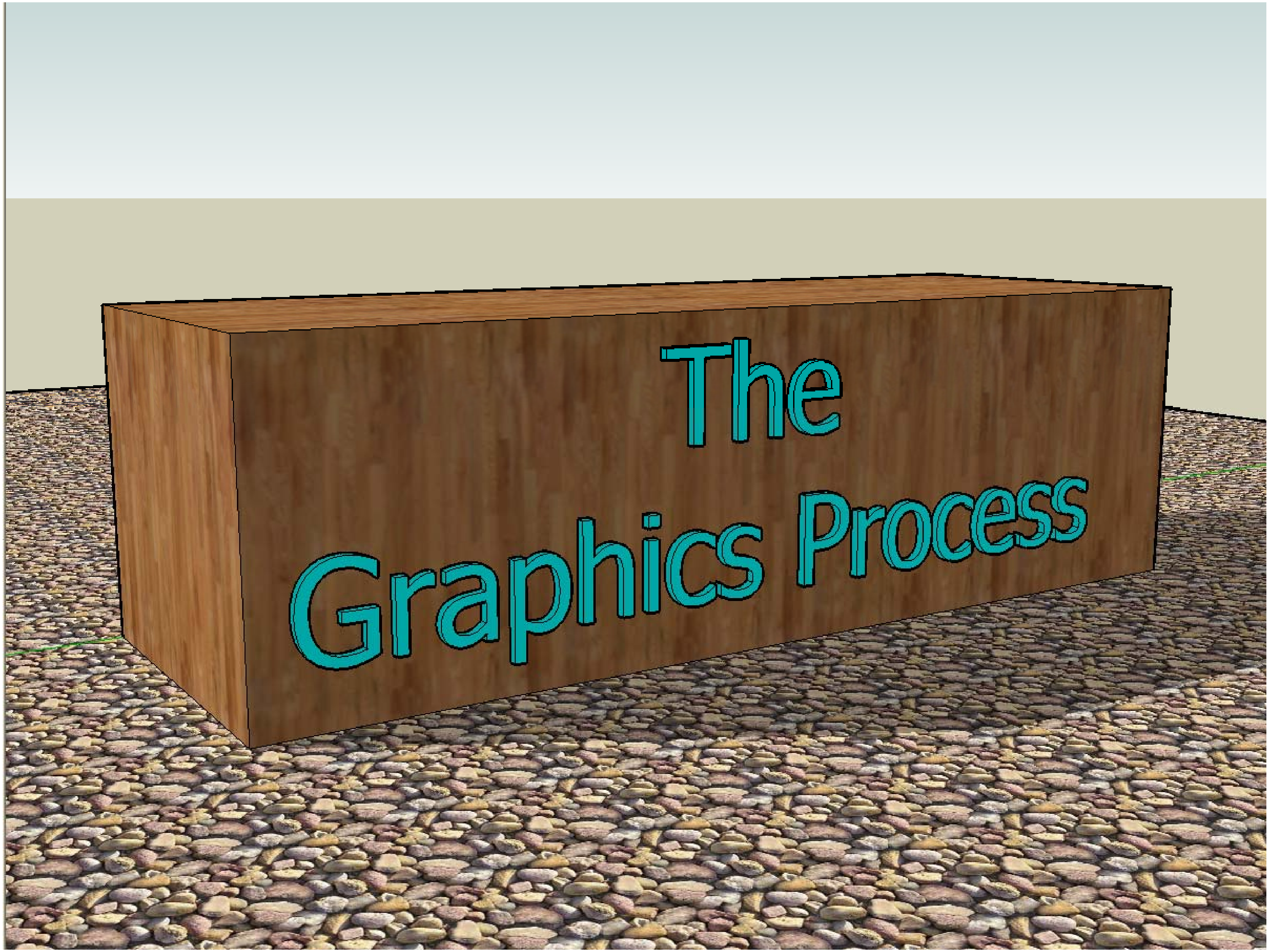
Sorted by booth number

	A	B	C	D	E
1	3D Consortium	1031		205	Digital Domain Institute
2	3D Systems	216		210	IEEE Computer Society
3	3D3 Solutions	1114		212	Blender Institute
4	3Dconnexion, Inc.	1013		216	3D Systems
5	4d View Solutions	745		217	DAZ 3D
6	Addison-Wesley/Pearson	919		225	RenderCloud
7	AMD	709		231	CyberGlove Systems
8	ARM	550		234	Web3D Consortium
9	Autodesk, Inc.	500		237	Point Grey Research, Inc.
10	Blender Institute	212		245	Objet Geometries Inc.
11	Canon Inc.	351		301	Pixologic, Inc.
12	Christie Digital Systems	1123		304	John Wiley & Sons, Inc.
13	Computer Graphics World	807		317	Intel Corporation
14	CRC Press/AK Peters	929		322	Stratasys 3D Printers & Production Systems
15	CyberGlove Systems	231		329	King Abdullah University of Science and Technology
16	DAZ 3D	217		334	DigiPen Institute of Technology
17	DigiPen Institute of Technology	334		335	Esri
18	Digital Domain Institute	205		351	Canon Inc.
19	Esri	335		354	Z Corporation
20	Focal Press/Morgan Kaufmann	825		357	Unity Technologies
21	Fox Render Farm	1112		500	Autodesk, Inc.
22	IEEE Computer Society	210		550	ARM
23	Intel Corporation	317		559	Khronos Demos
24	John Wiley & Sons, Inc.	304		610	OptiTrack
25	JourneyEd	1018		634	NVIDIA Corporation
26	Khronos Demos	559		644	Pixar Animation Studios
27	Khronos Education	858		709	AMD
28	Khronos Group	759		735	NVIDIA Corporation
29	Khronos Theater	758		745	4d View Solutions
30	King Abdullah University of Science and Technology	329		745	SolidAnim
31	NVIDIA Corporation	634		758	Khronos Theater
32	NVIDIA Corporation	735		759	Khronos Group
33	Objet Geometries Inc.	245		807	Computer Graphics World
34	OptiTrack	610		819	Wacom Technology Services, Corp.
35	Pixar Animation Studios	644		825	Focal Press/Morgan Kaufmann
36	Pixologic, Inc.	301		831	PNY Technologies
37	PNY Technologies	831		858	Khronos Education
38	Point Grey Research, Inc.	237		909	Zygote Media Group, Inc.
39	Qt Commercial, Digia	1000		913	Springer
40	RenderCloud	225		919	Addison-Wesley/Pearson
41	SolidAnim	745		929	CRC Press/AK Peters
42	Springer	913		1000	Qt Commercial, Digia
43	StereoArt	1020		1013	3Dconnexion, Inc.
44	Stratasys 3D Printers & Production Systems	322		1018	JourneyEd
45	The Foundry	1019		1019	The Foundry
46	Unity Technologies	357		1020	StereoArt
47	Wacom Technology Services, Corp.	819		1031	3D Consortium
48	Web3D Consortium	234		1051	Western Digital
49	Western Digital	1051		1112	Fox Render Farm
50	WorldViz	1161		1114	3D3 Solutions
51	Z Corporation	354		1123	Christie Digital Systems
52	Zygote Media Group, Inc.	909		1161	WorldViz
53					

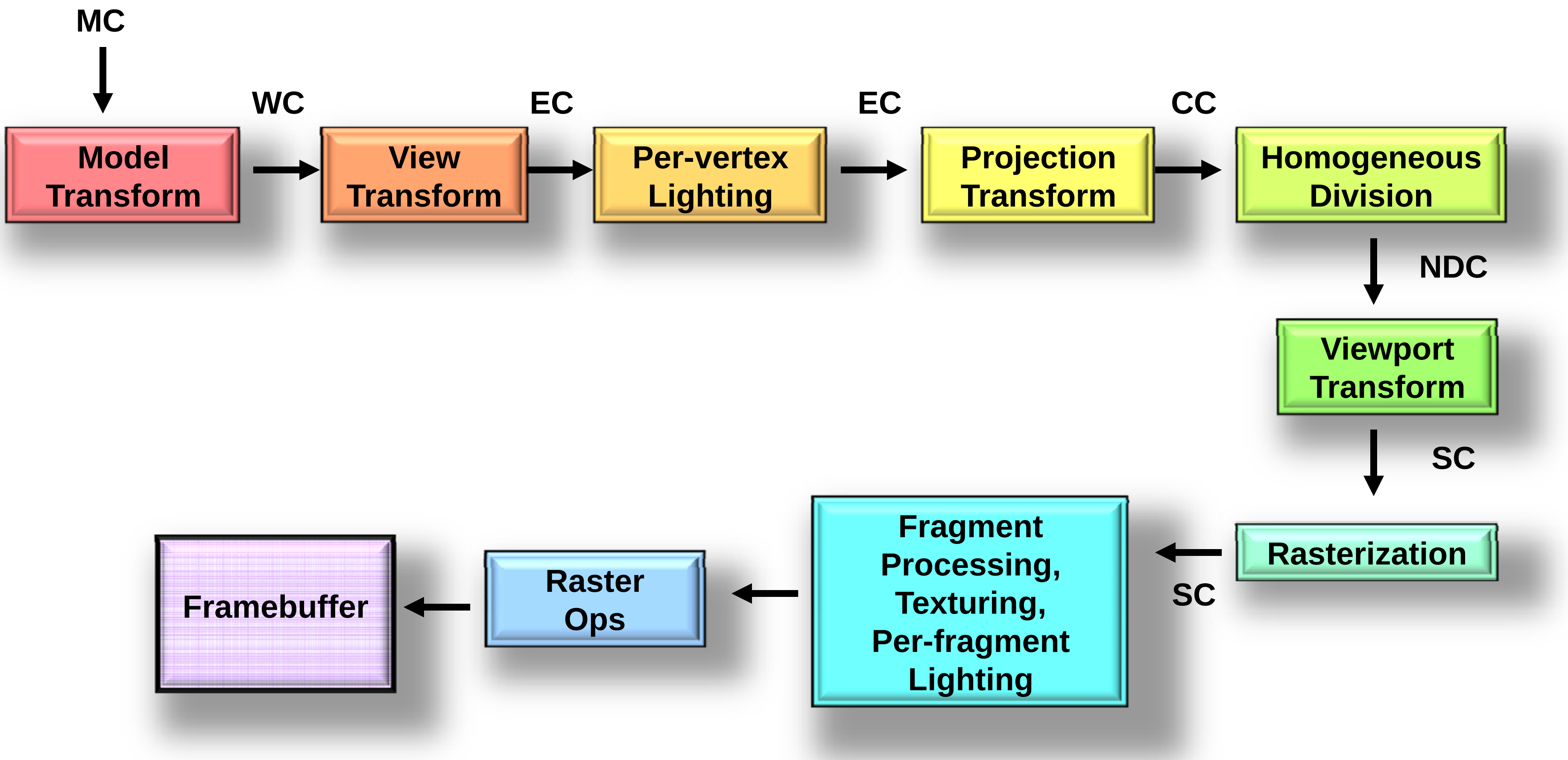
Exhibition Strategy



Oregon State
Computer Graphics


The
Graphics Process

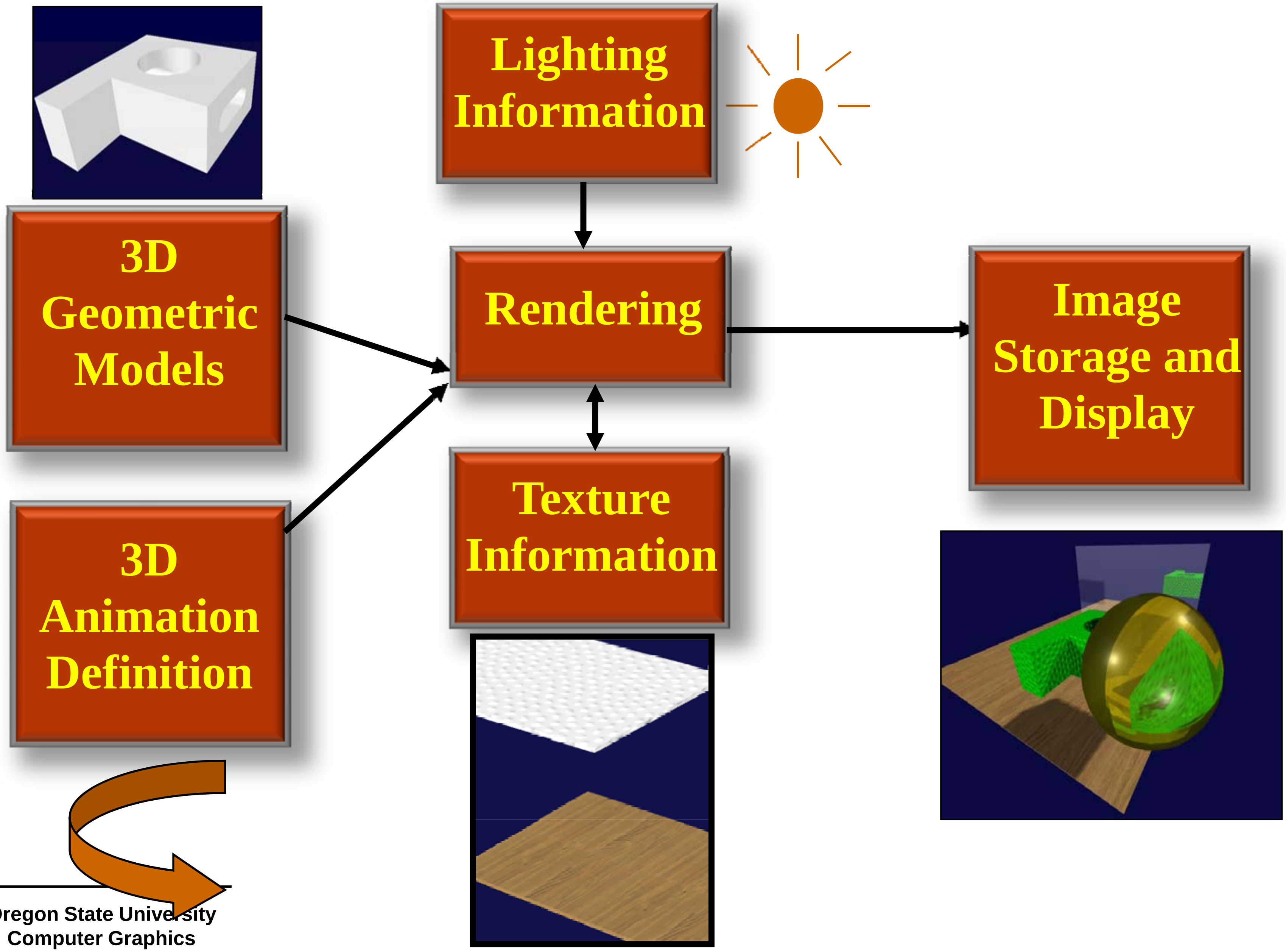
The Basic Computer Graphics Pipeline



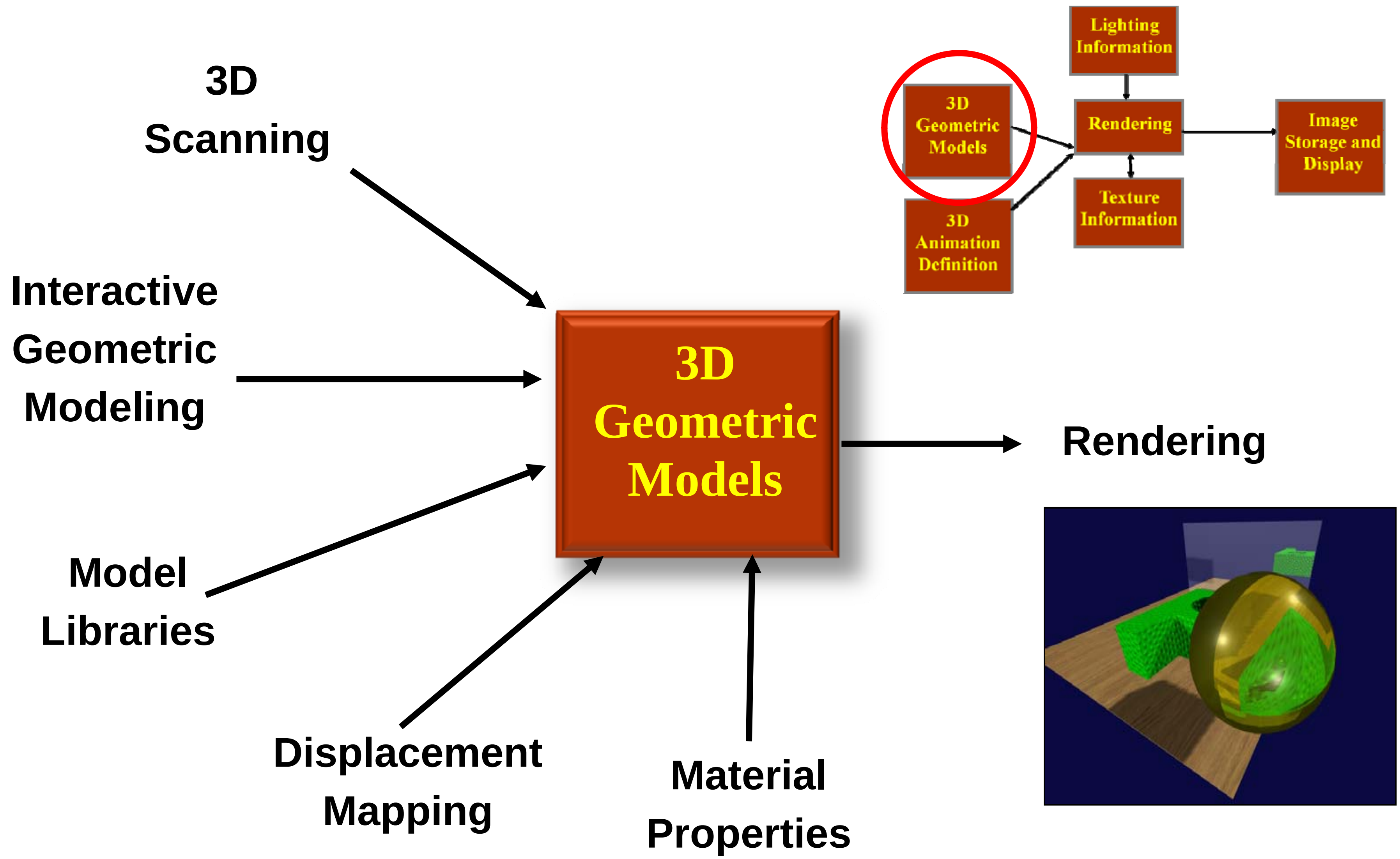
MC = Model Coordinates
WC = World Coordinates
EC = Eye Coordinates
CC = Clip Coordinates
NDC = Normalized Device Coordinates
SC = Screen Coordinates

mjb – June 12. 2014

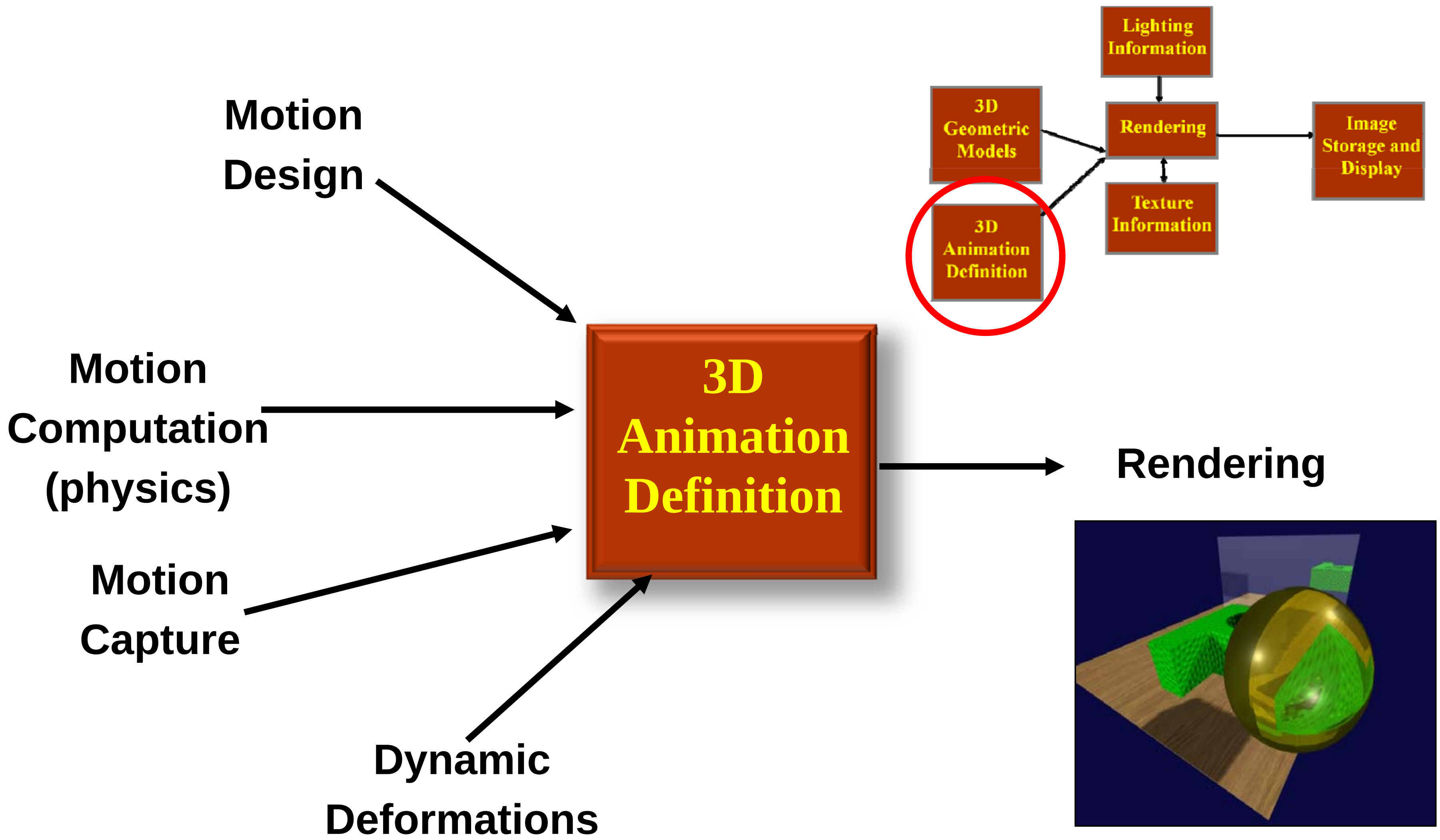
The Graphics Process



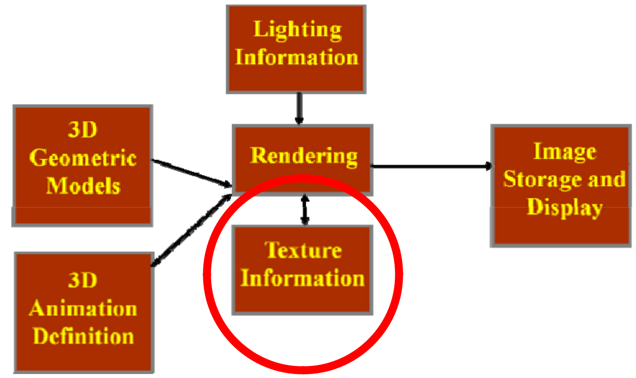
The Graphics Process: Geometric Modeling



The Graphics Process: 3D Animation



The Graphics Process: Texturing



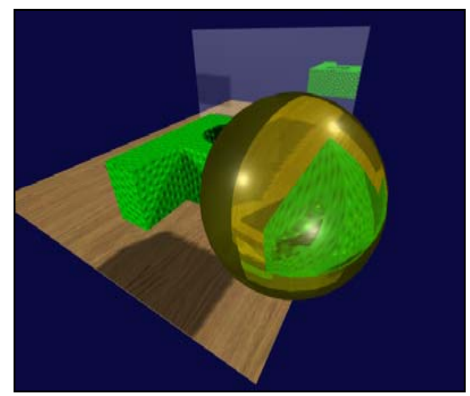
Scanned Image Textures

Procedural (computed) Textures

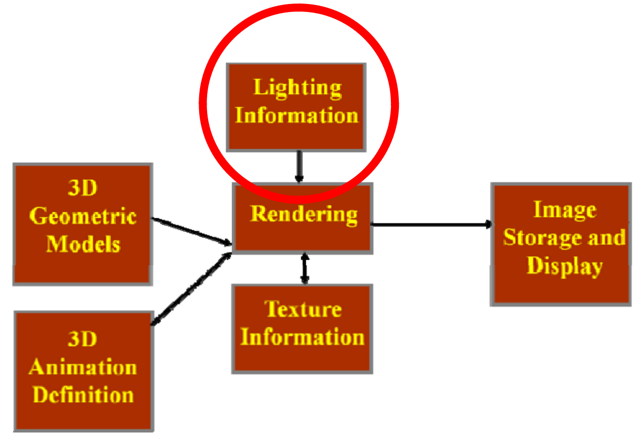
Painted Textures

Texture Information

Rendering



The Graphics Process: Lighting



Lighting Types (point, directional, spot)

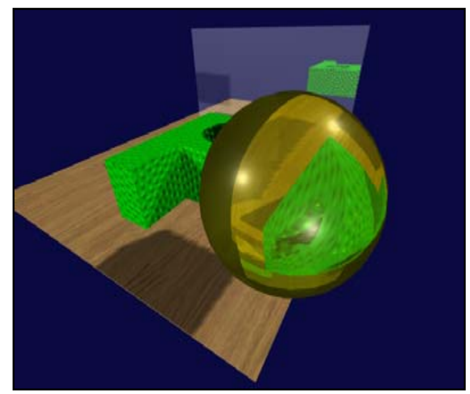
Light Positions

Lighting Information

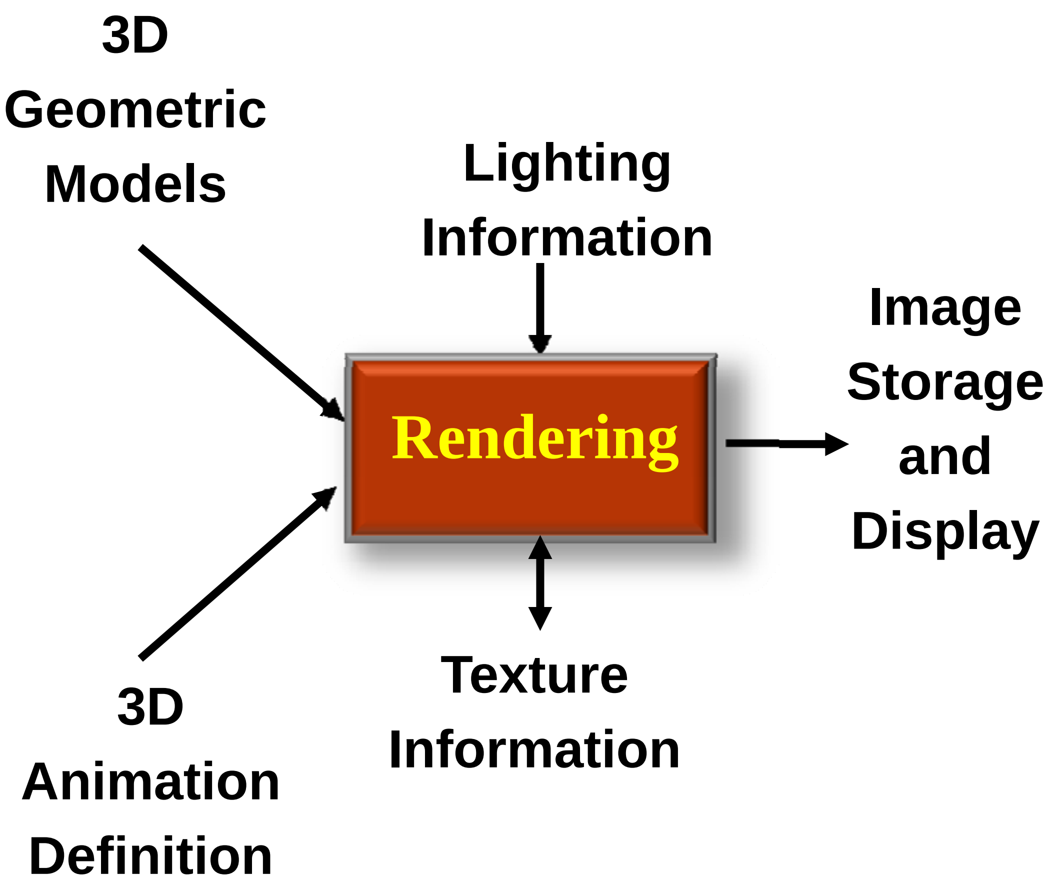
Rendering

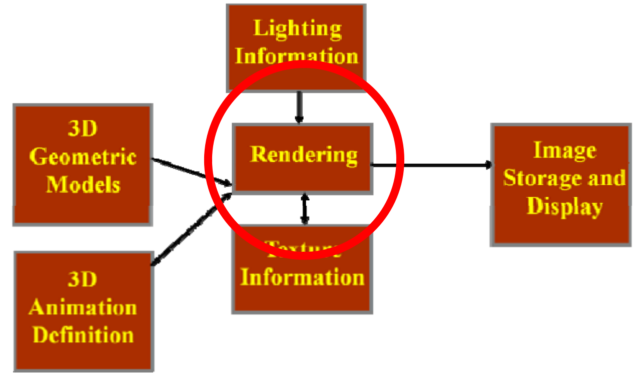
Light Colors

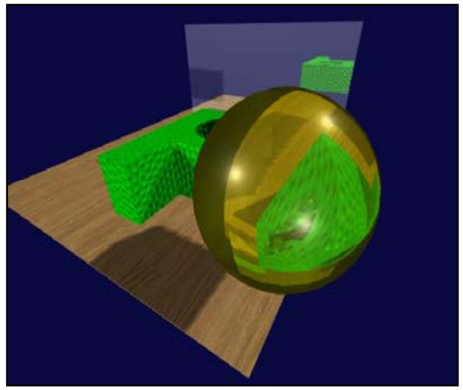
Light Intensities



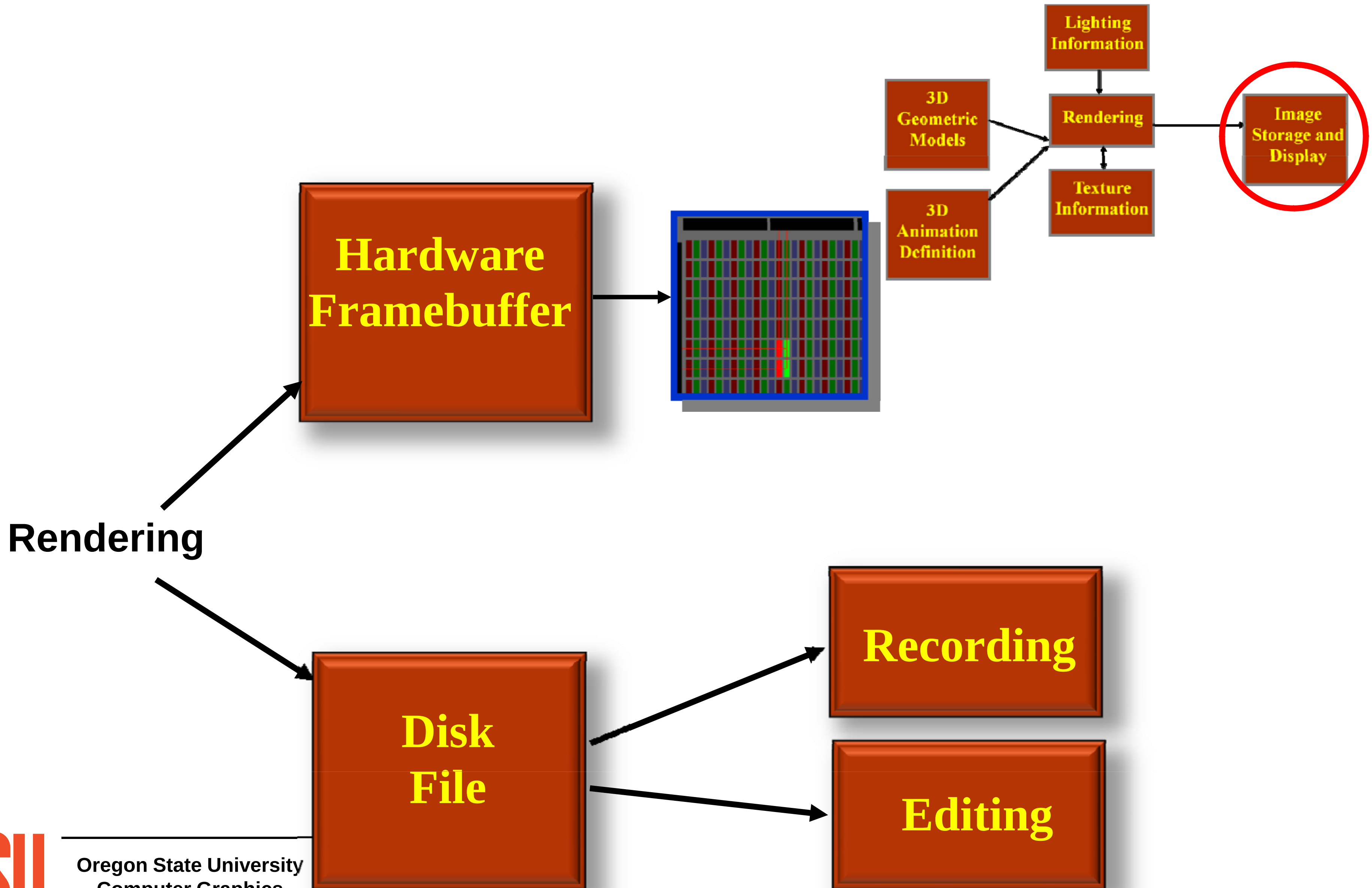
The Graphics Process: Rendering



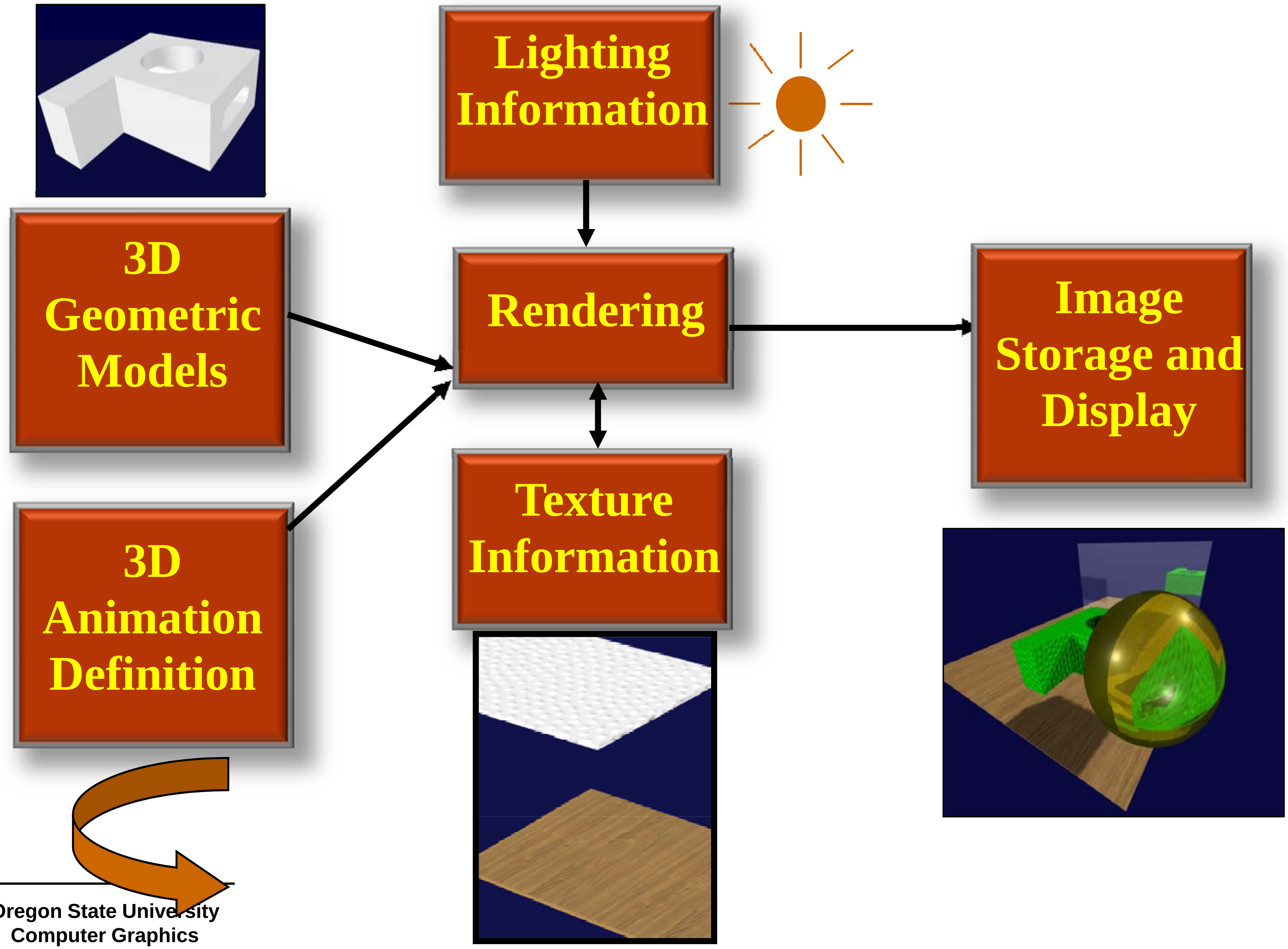


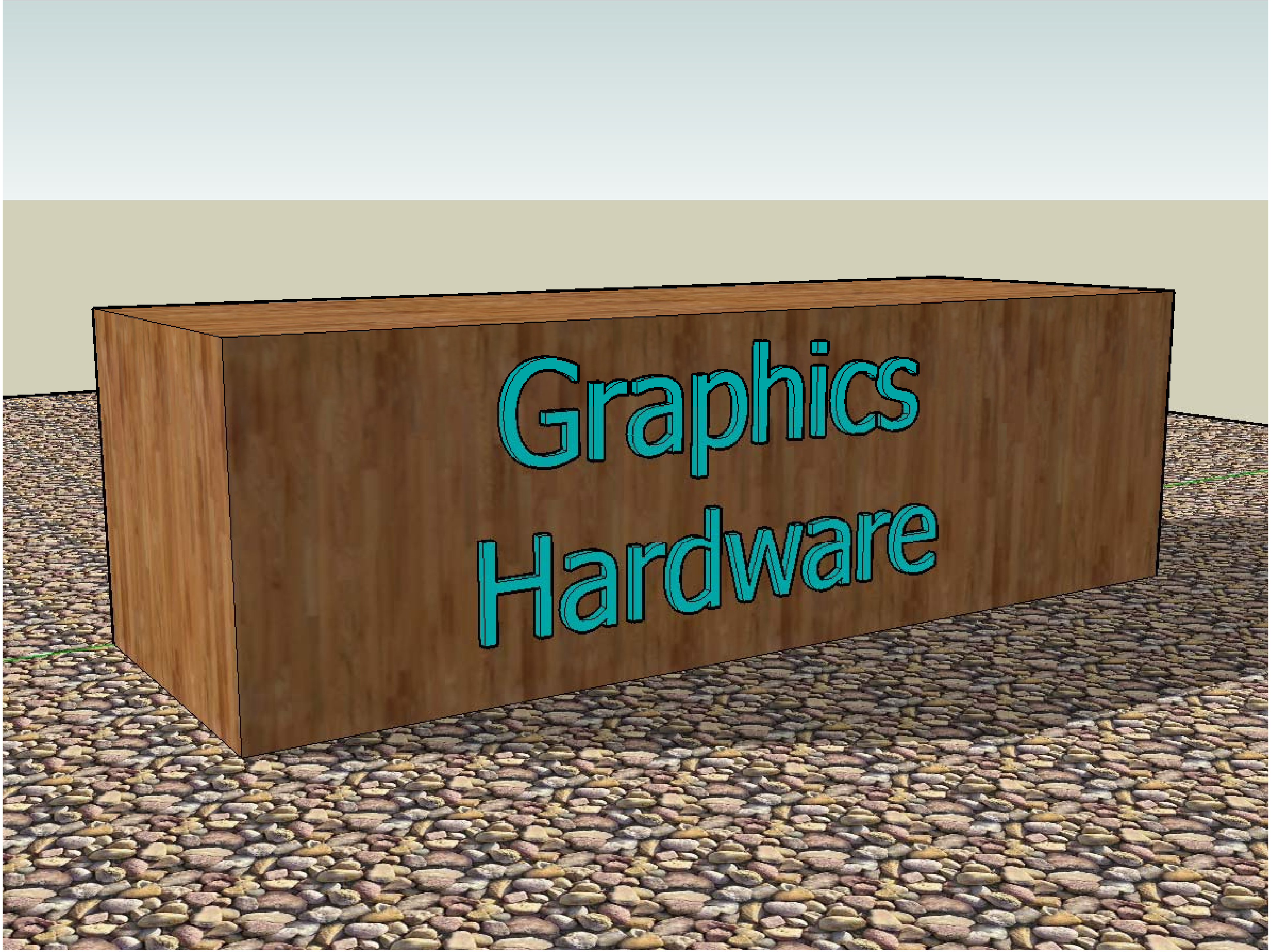


The Graphics Process:
Image Storage and Display

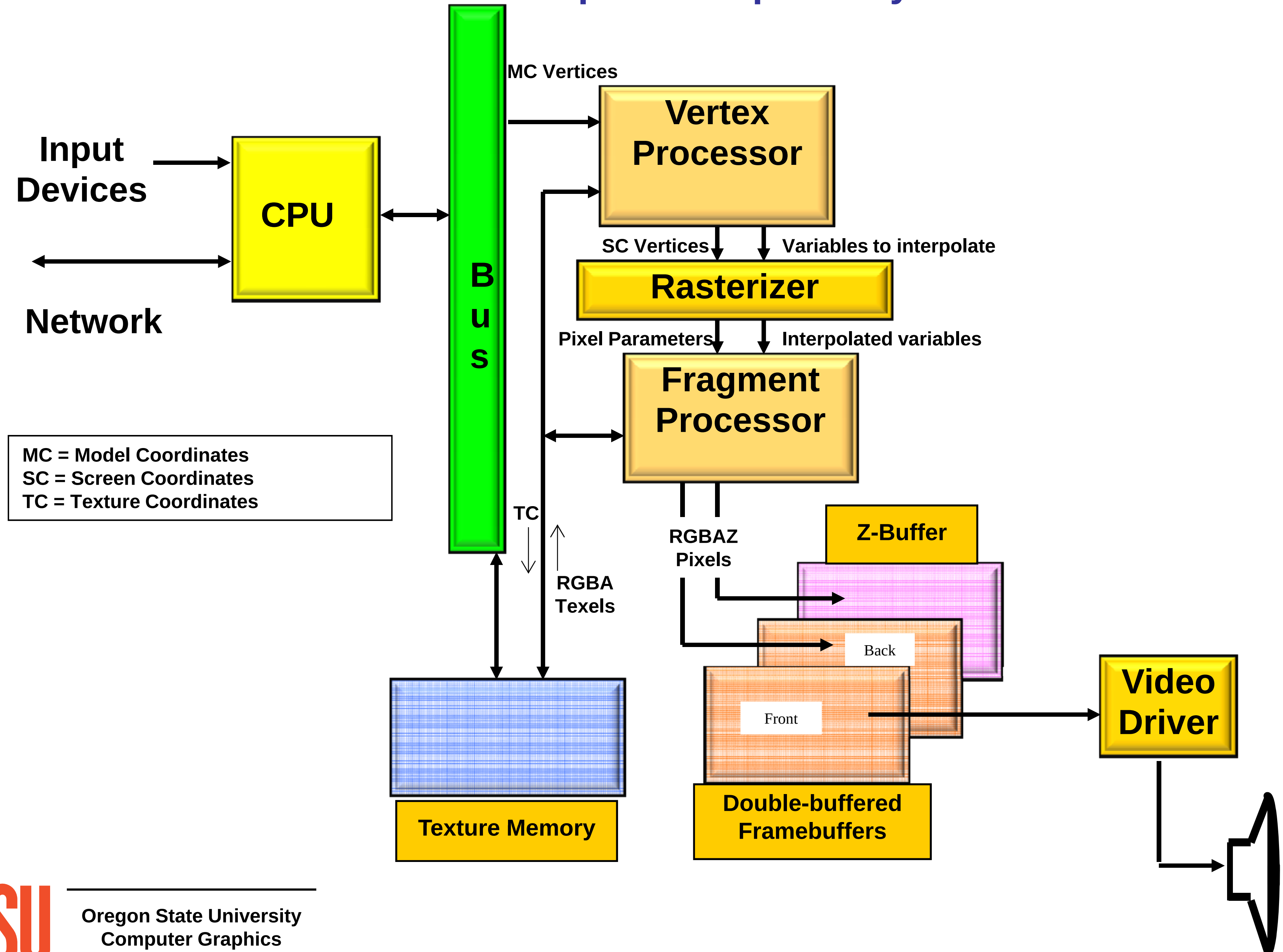


The Graphics Process; Summary

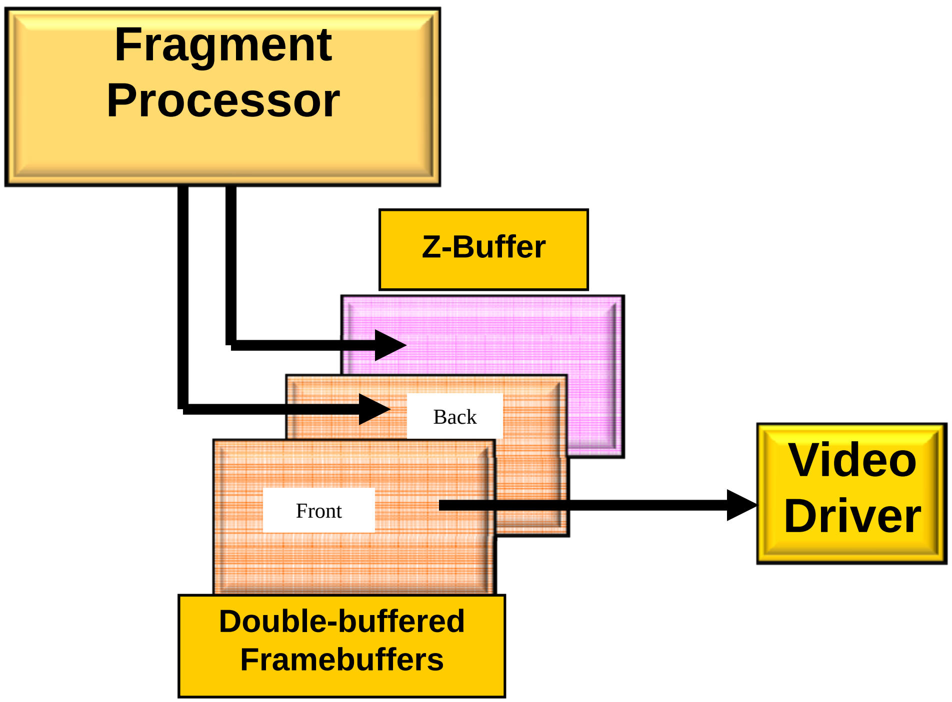


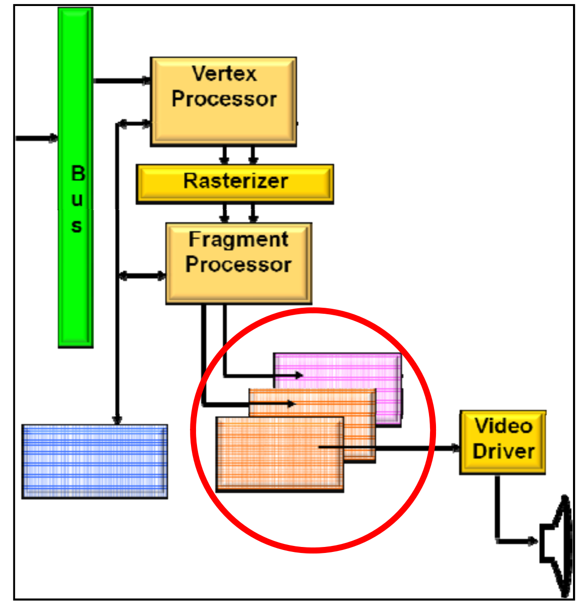

Graphics
Hardware

Generic Computer Graphics System



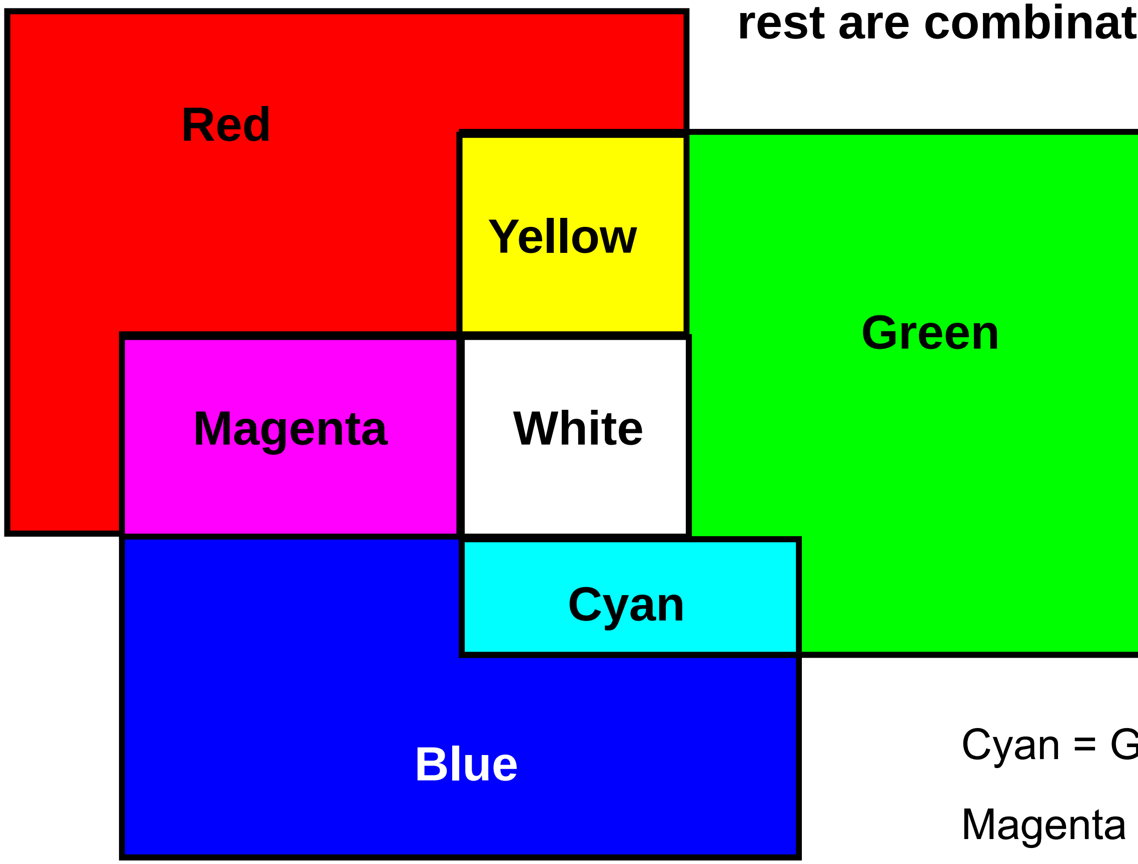
The Framebuffer





The Framebuffer Uses Additive Colors (RGB)

Red, Green, and Blue are provided. The rest are combinations of those three.



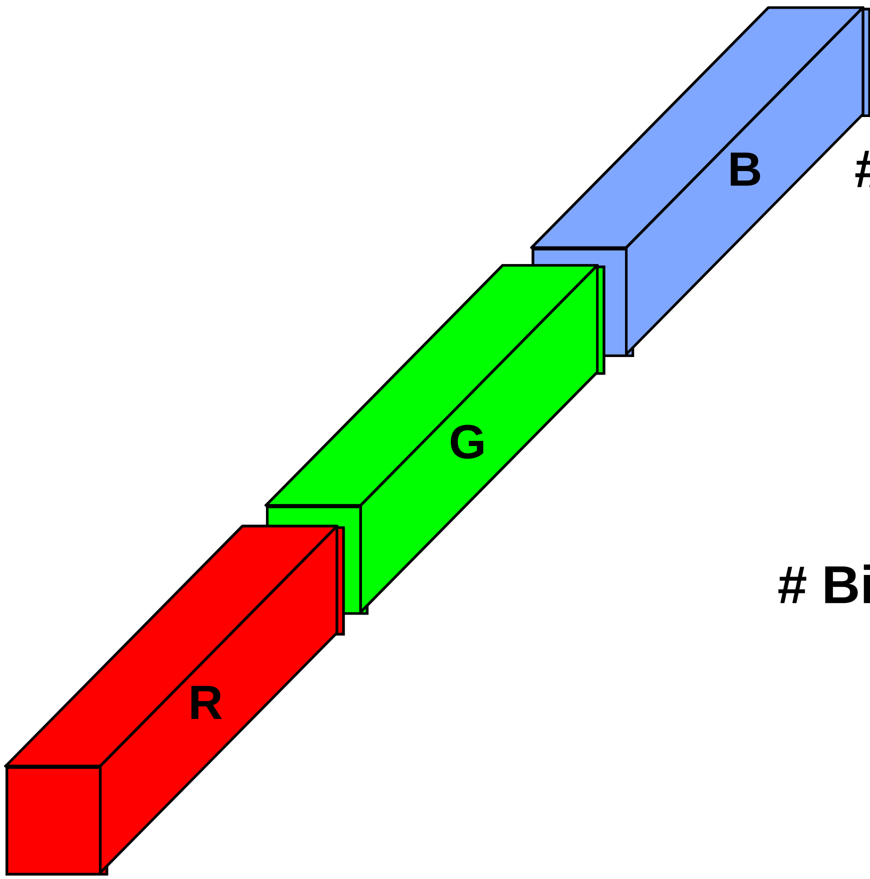
Cyan = Green + Blue

Magenta = Red + Blue

Yellow = Red + Green

White = Red + Green + Blue

The Framebuffer:
Integer Color Storage

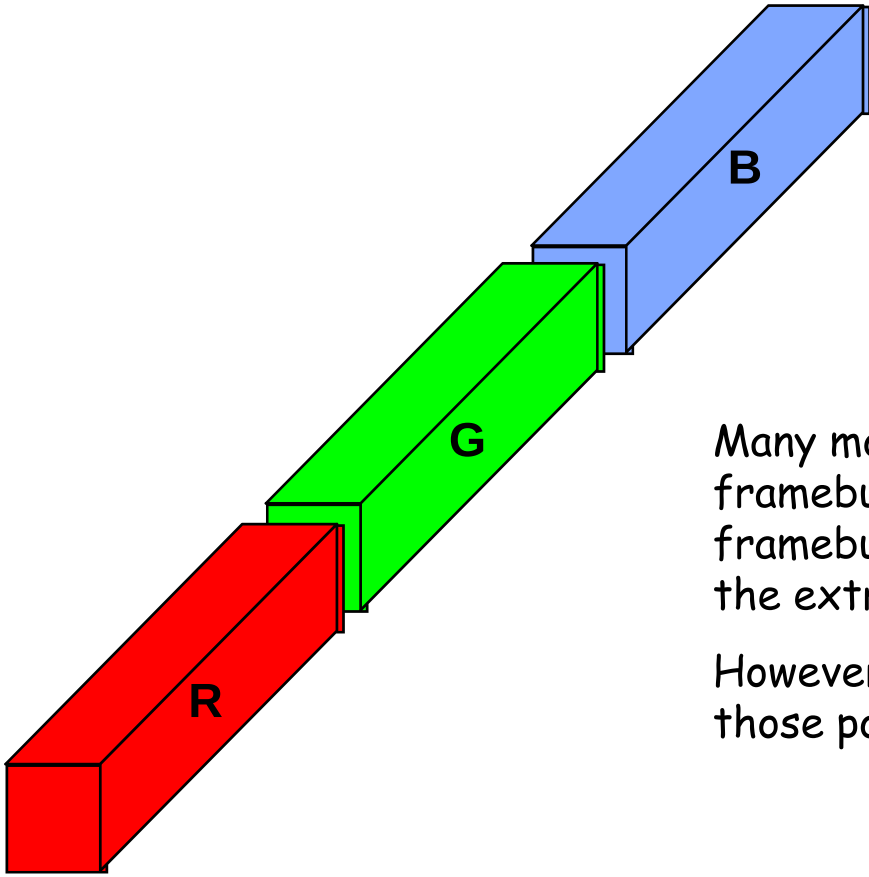


# Bits/color	# Intensities per color
8	2^8 = 256
10	2^{10} = 1024
12	2^{12} = 4096

# Bits/pixel	Total colors:
24	2^{24} = 16.7 M
30	2^{30} = 1 B
36	2^{36} = 69 B

The Framebuffer:
Floating Point Color Storage

- ***16- or 32-bit floating point for each color component***



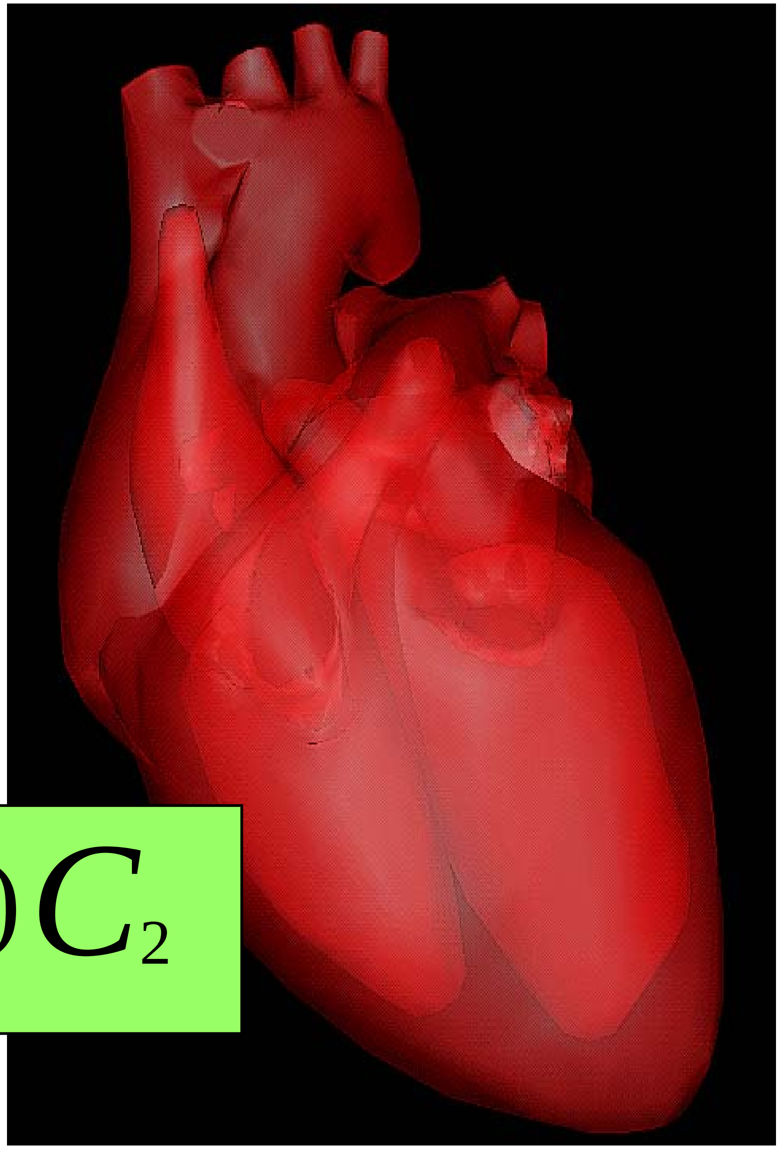
Why so much?

Many modern algorithms do arithmetic on the framebuffer color components, or treat the framebuffer color components as data. They need the extra precision during the arithmetic.

However, the display system cannot display all of those possible colors.

The Framebuffer

- ***Alpha*** values
  - Transparency per pixel
    α = 0. is invisible
    α = 1. is opaque
  - Represented in 8-32 bits
    (integer or floating point)
  - Alpha blending equation:

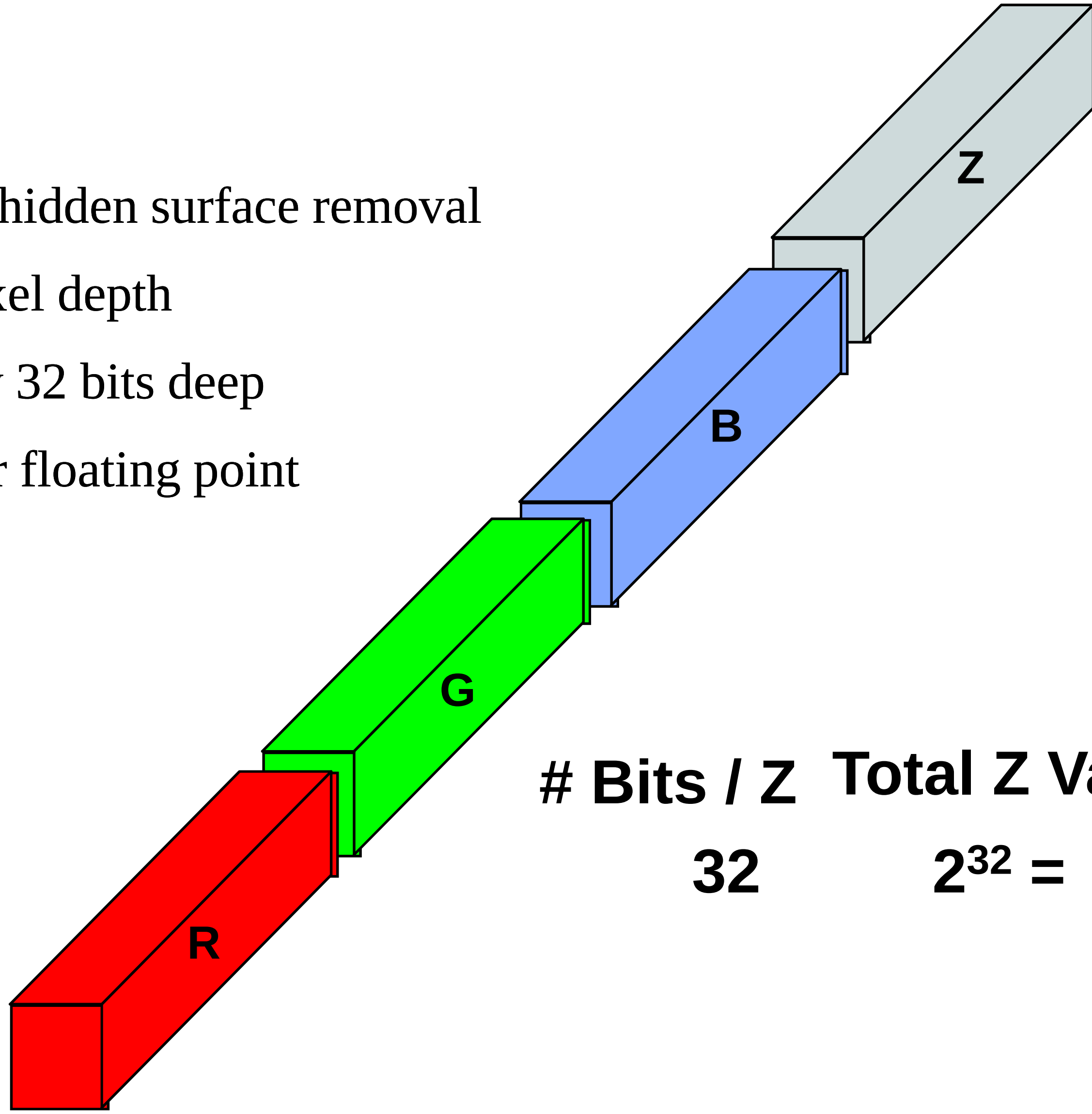


$$Color = \alpha C_1 + (1 - \alpha) C_2$$

$$0.0 \leq \alpha \leq 1.0$$

The Framebuffer

- ***Z-buffer***
  - Used for hidden surface removal
  - Holds pixel depth
  - Typically 32 bits deep
  - Integer or floating point

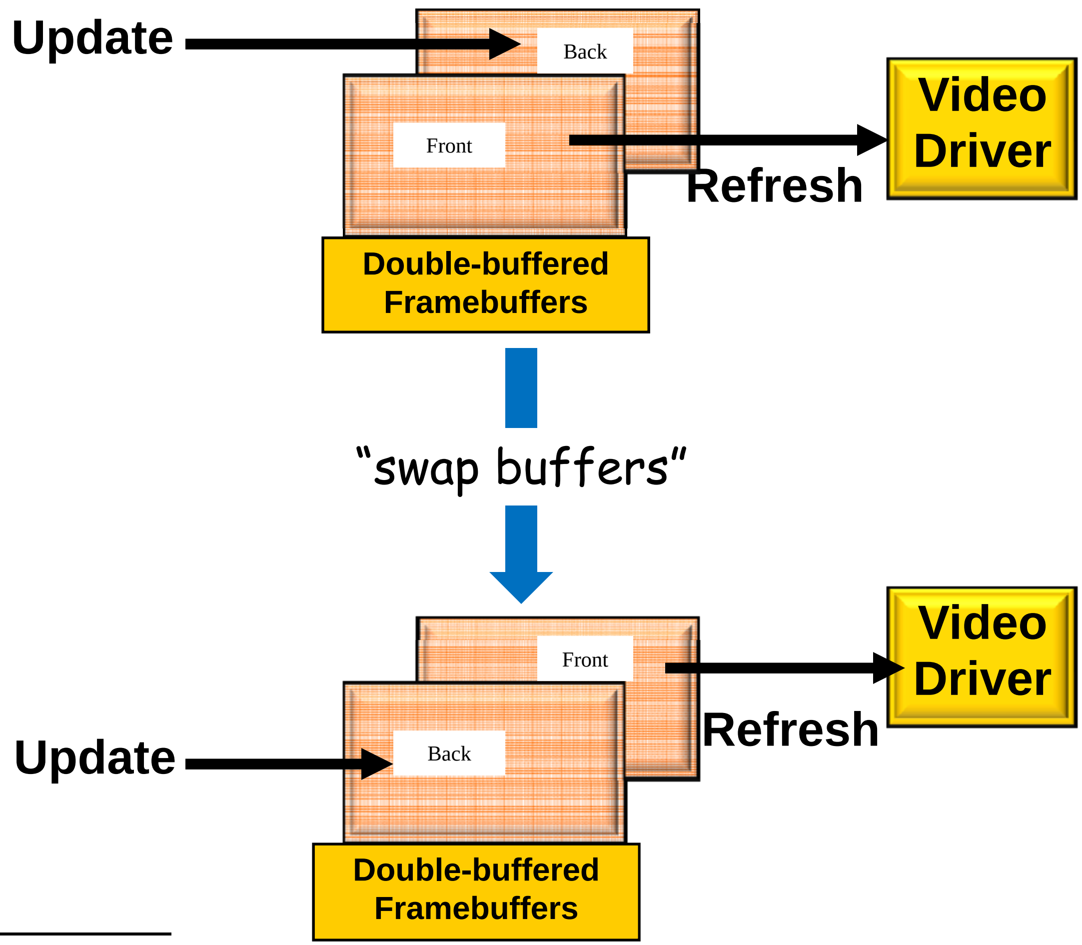


 # Bits / Z **Total Z Values:**

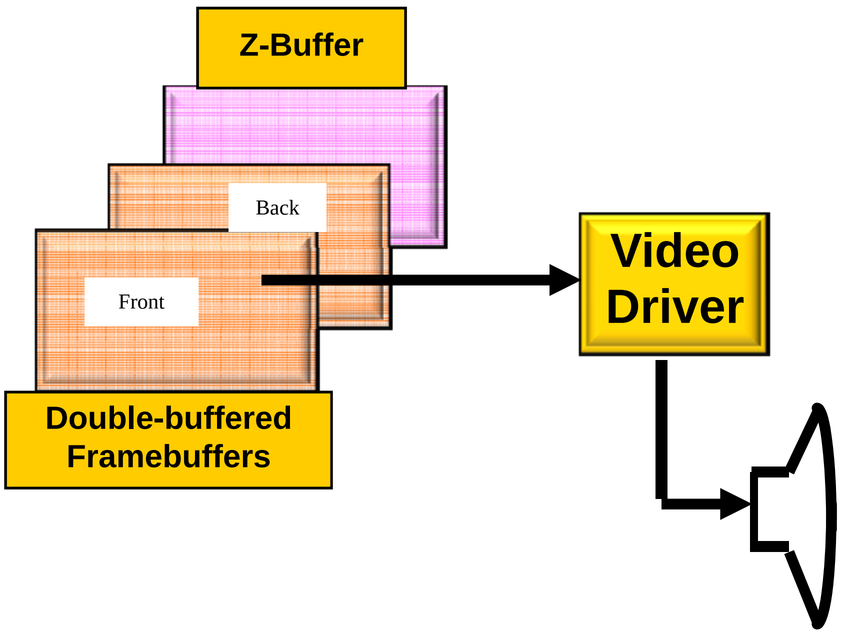
32 $\mathbf{2^{32}}$ **= 4 B**

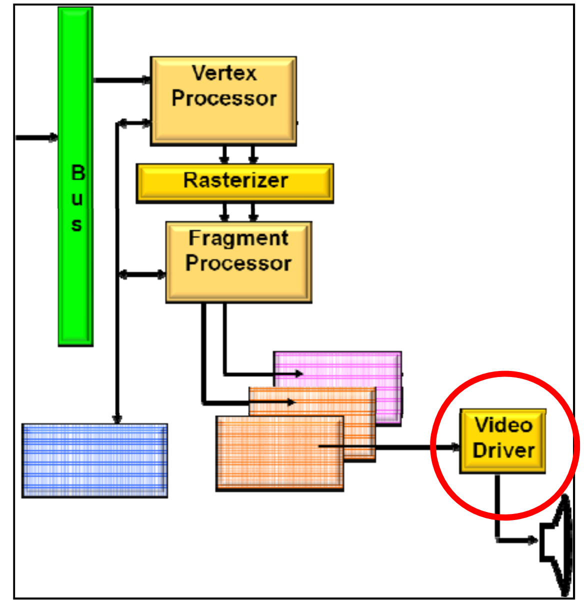
The Framebuffer

Double-buffering: Don't let the viewer see *any* of the scene until the entire scene is drawn



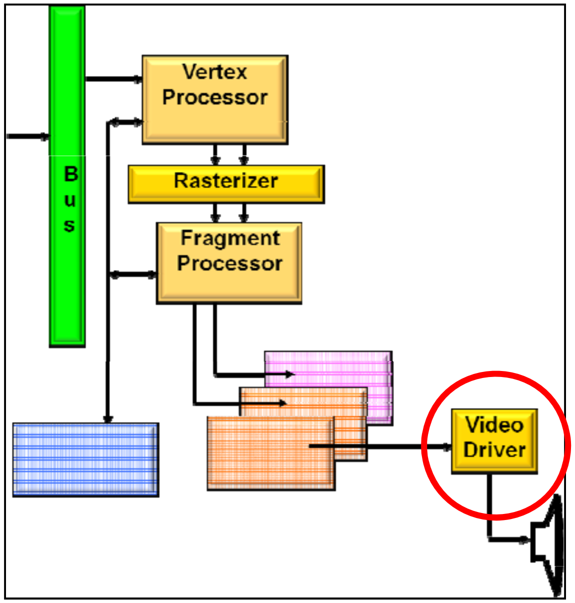
The Video Driver



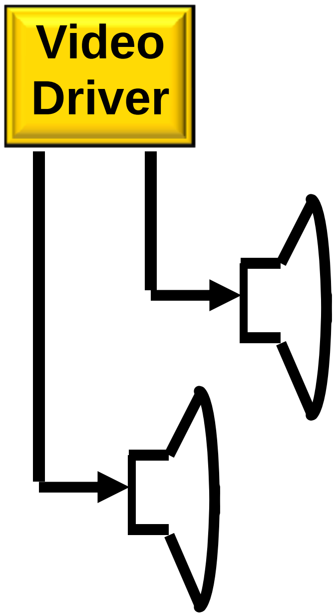


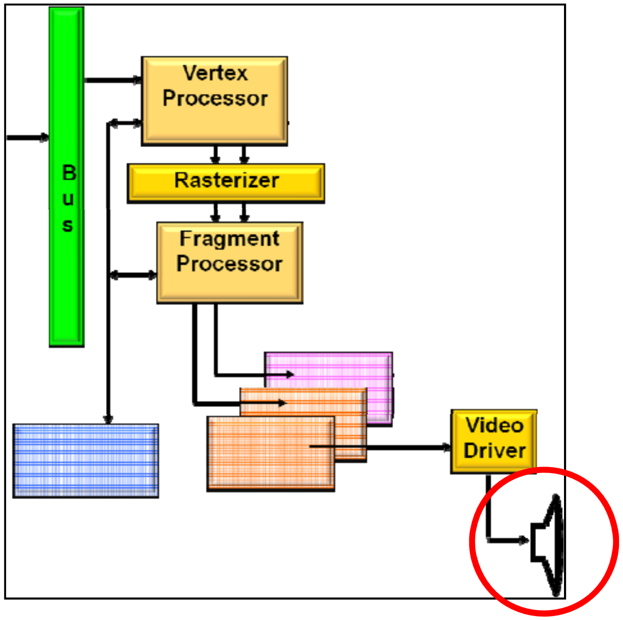
The Video Driver

- N ***refreshes*/second** (N is usually between 50 and 100)
- Framebuffer contains the R,G,B that define the color at each pixel
- Cursor
  - Appearance is stored near the video driver in a “mini-framebuffer”
  - x,y is given by the CPU
- Video input



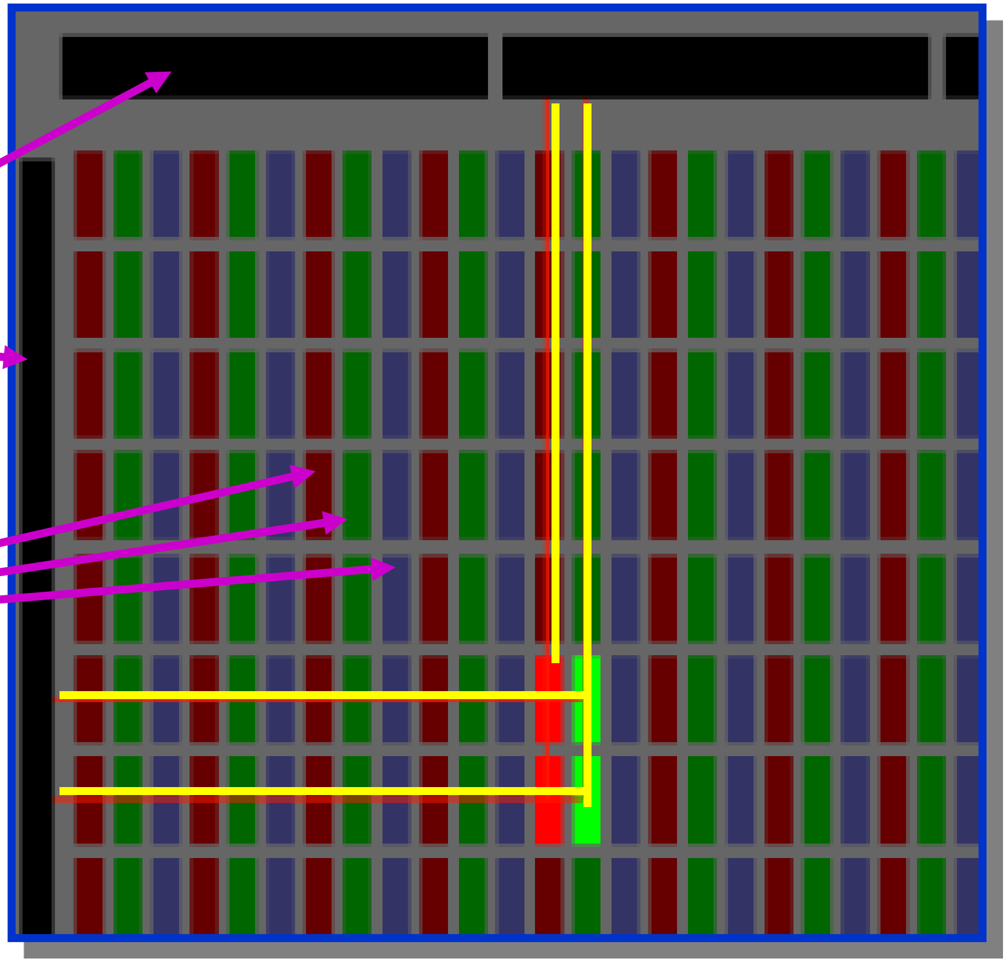
The Computer Graphics Monitor(s)





Displaying Color on a Computer Graphics LCD Monitor

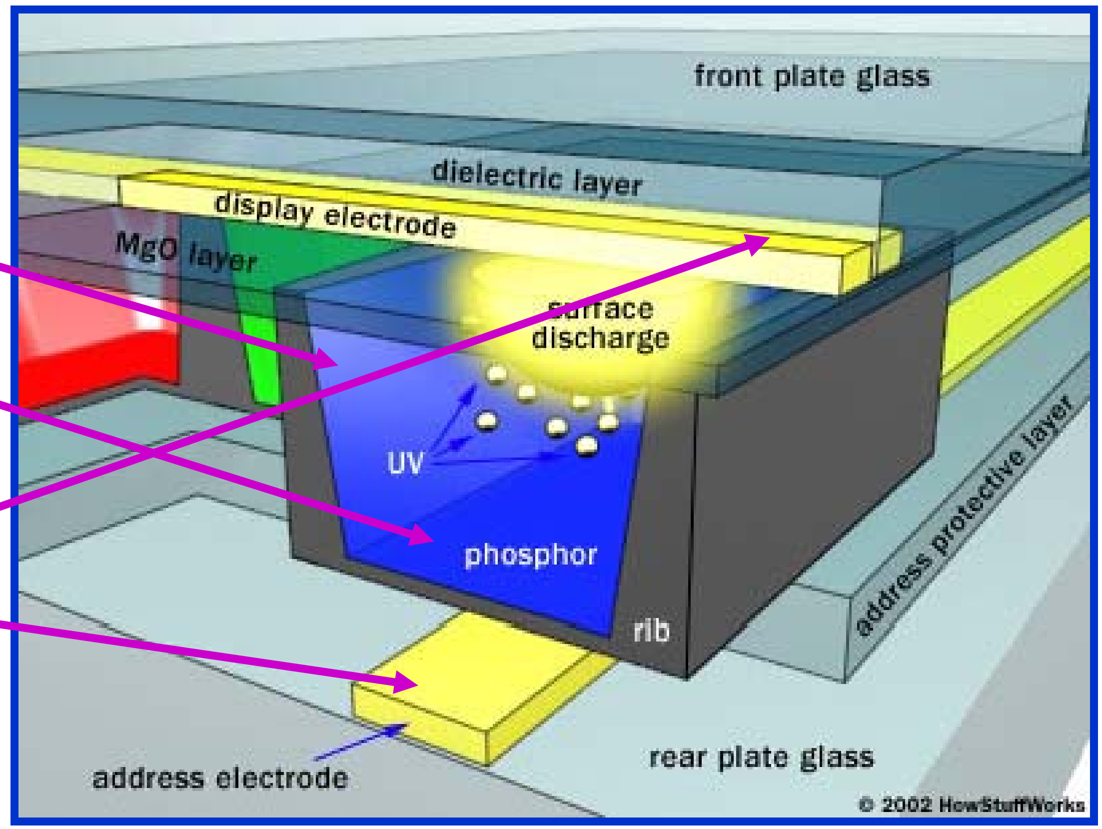
- Grid of electrodes
- Color filters



Source: http://electronics.howstuffworks.com

Displaying Color on a Plasma Monitor

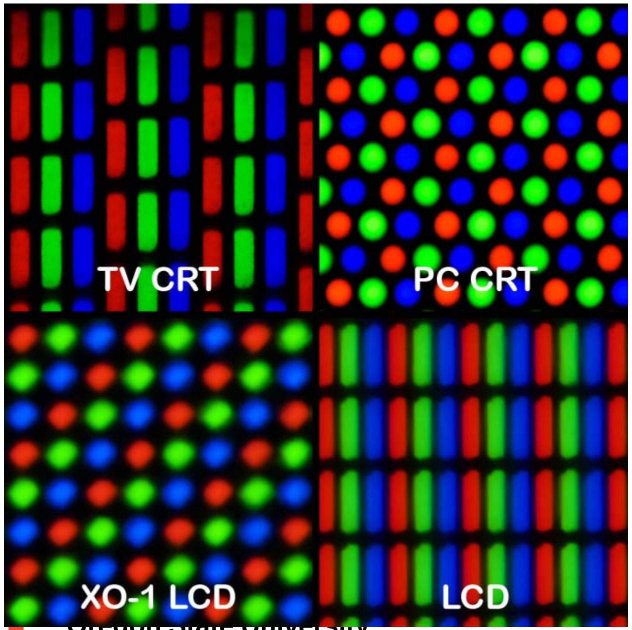
- **Gas cell**
- **Phosphor**
- **Grid of electrodes**



http://electronics.howstuffworks.com

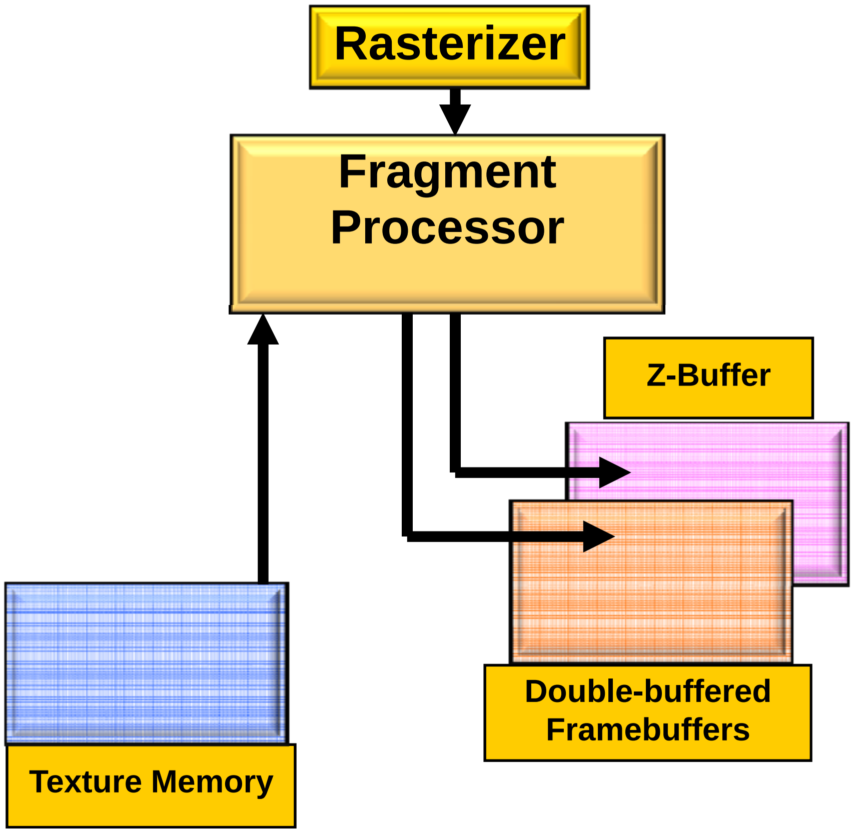
Display Resolution

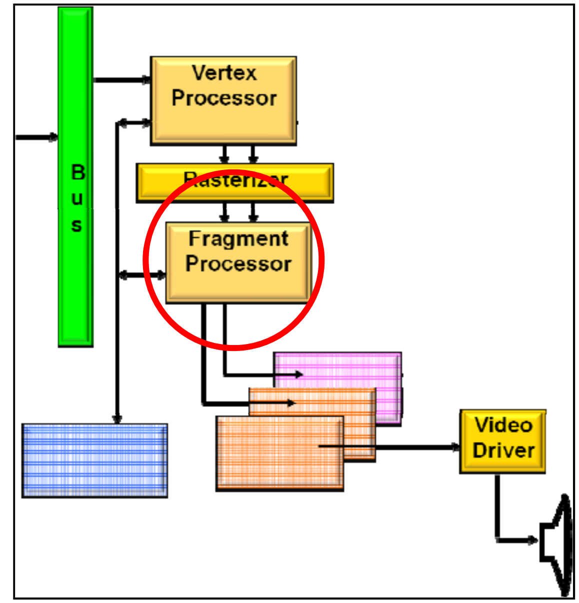
- ***Pixel*** resolutions (1280x1024, 1600x1200, 1920x1152 are common on the desktop)
- “4096” is 4096 x 2160
- LG’s new Ultra Widescreen is 3440 x 1440, 34”
- Human acuity: 1 arc-minute is achieved by viewing a 19" monitor with 1280x1024 resolution from a distance of ~40 inches



http://en.wikipedia.org/wiki/File:Pixel_geometry_01_Pengo.jpg

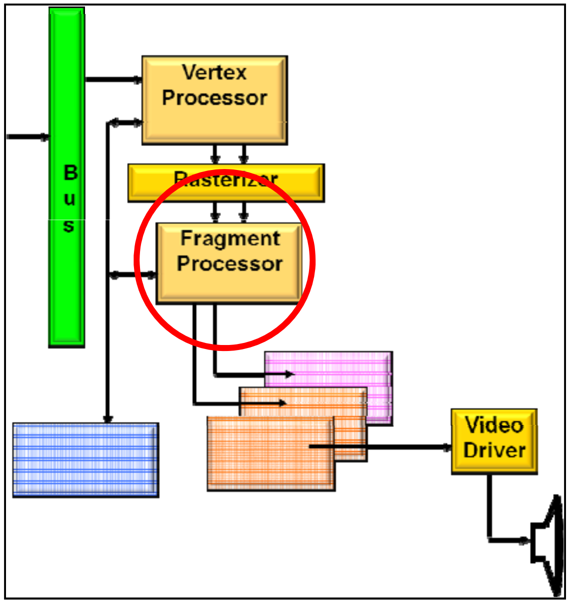
The Fragment Processor



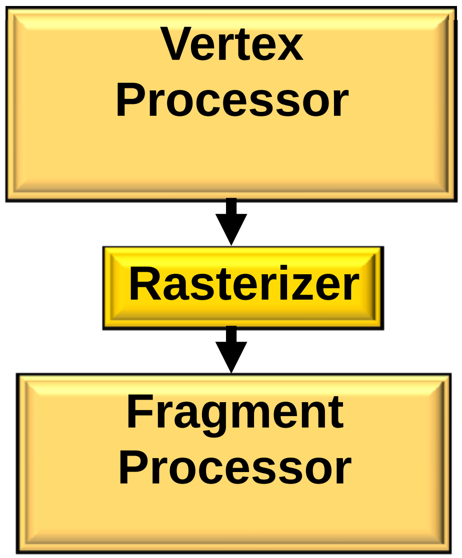


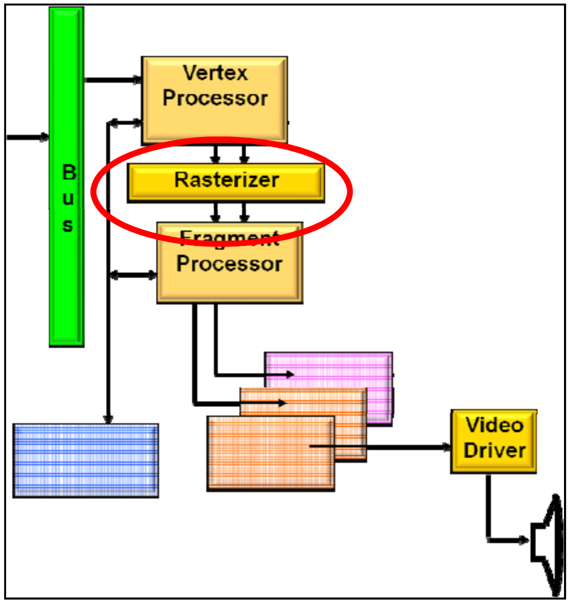
The Fragment Processor

- Takes in all information that describes this pixel
- Produces the RGBA for that pixel’s location in the framebuffer



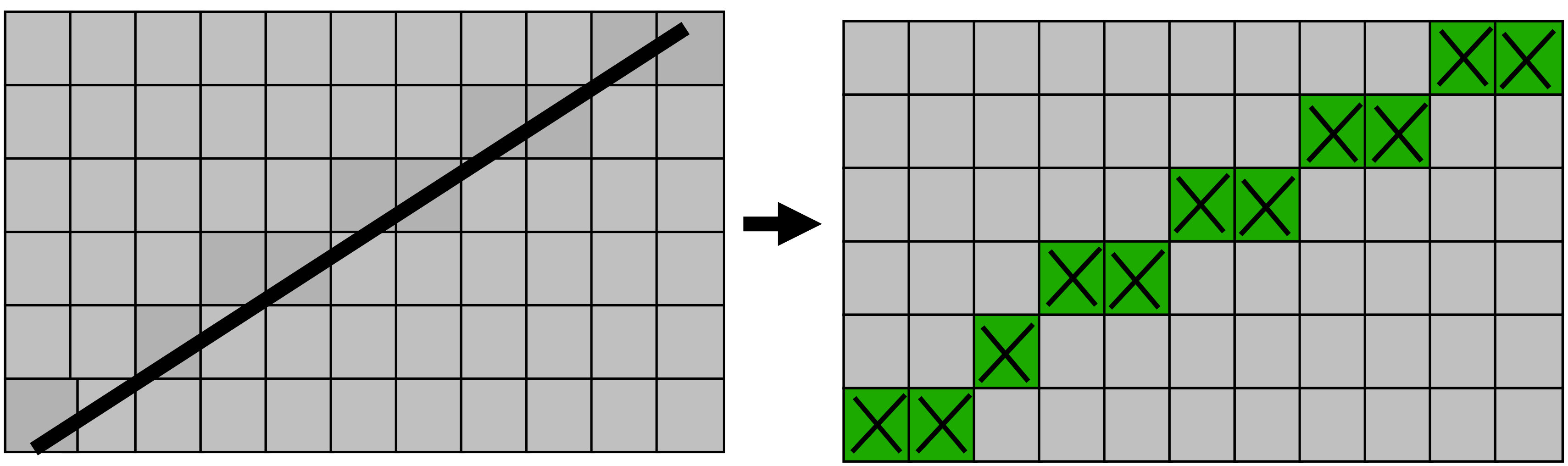
The Rasterizer



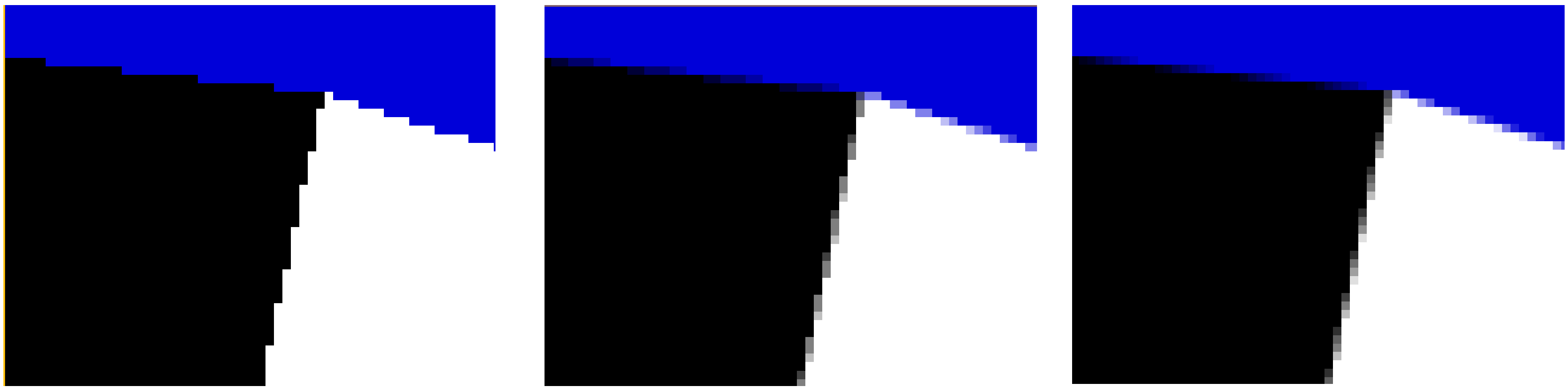


Rasterization

- Turn screen space vertex coordinates into pixels that make up lines and polygons
- A great place for custom electronics
- Anti-aliasing is often built-in



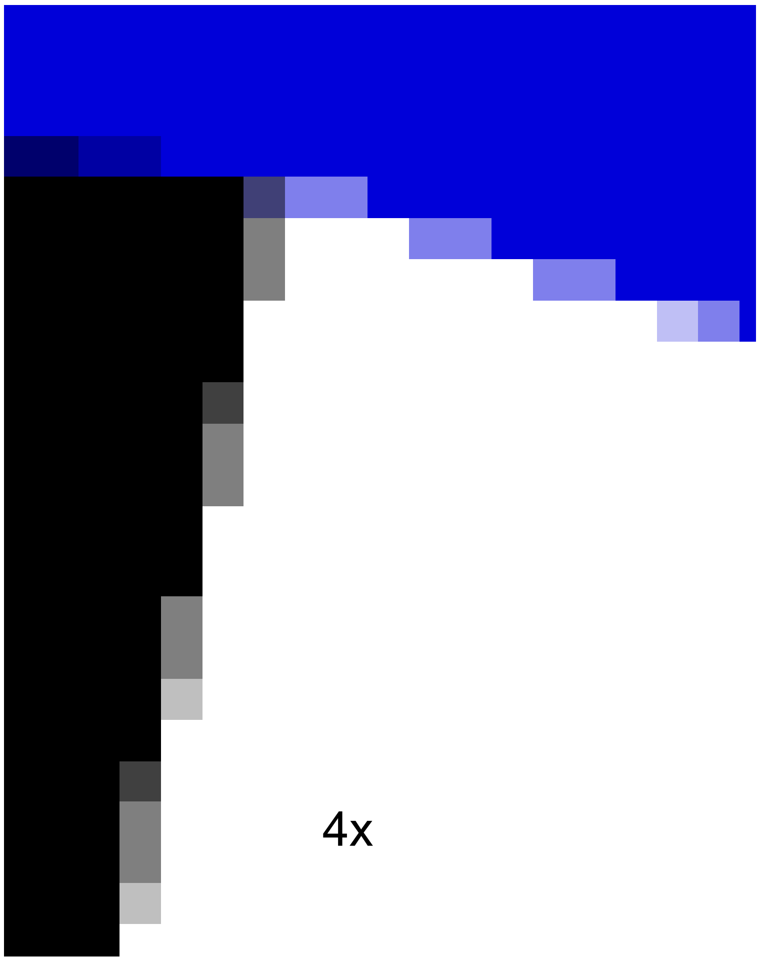
Anti-aliasing is Implemented by Oversampling within Each Pixel

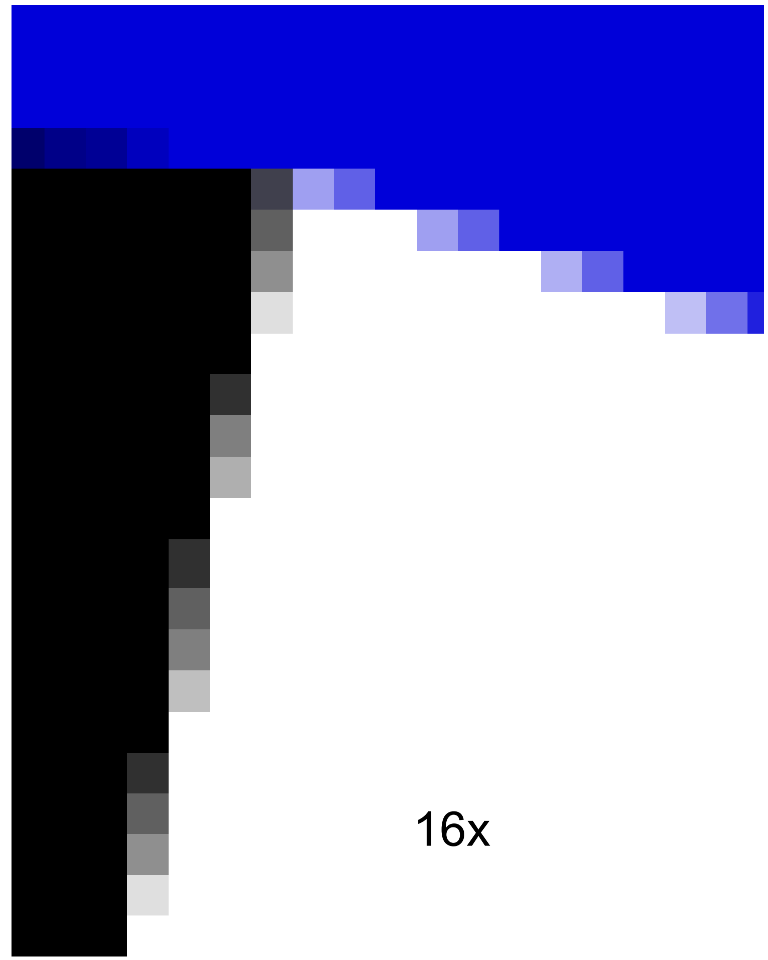


No AA 4x 16x

NVIDIA

Anti-aliasing is Implemented by Oversampling within Each Pixel

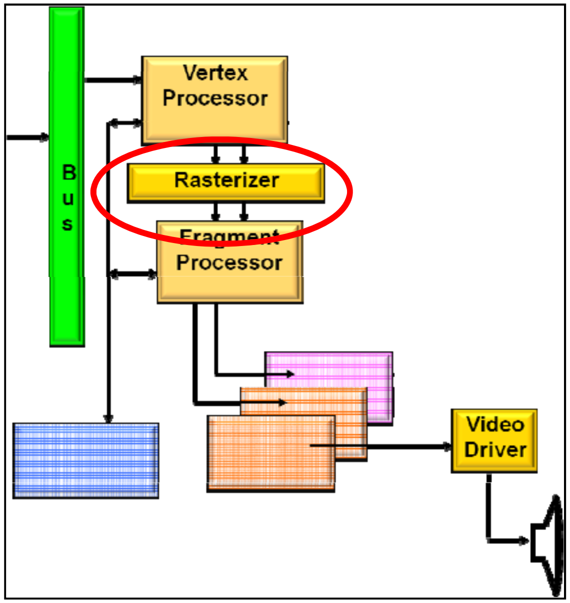




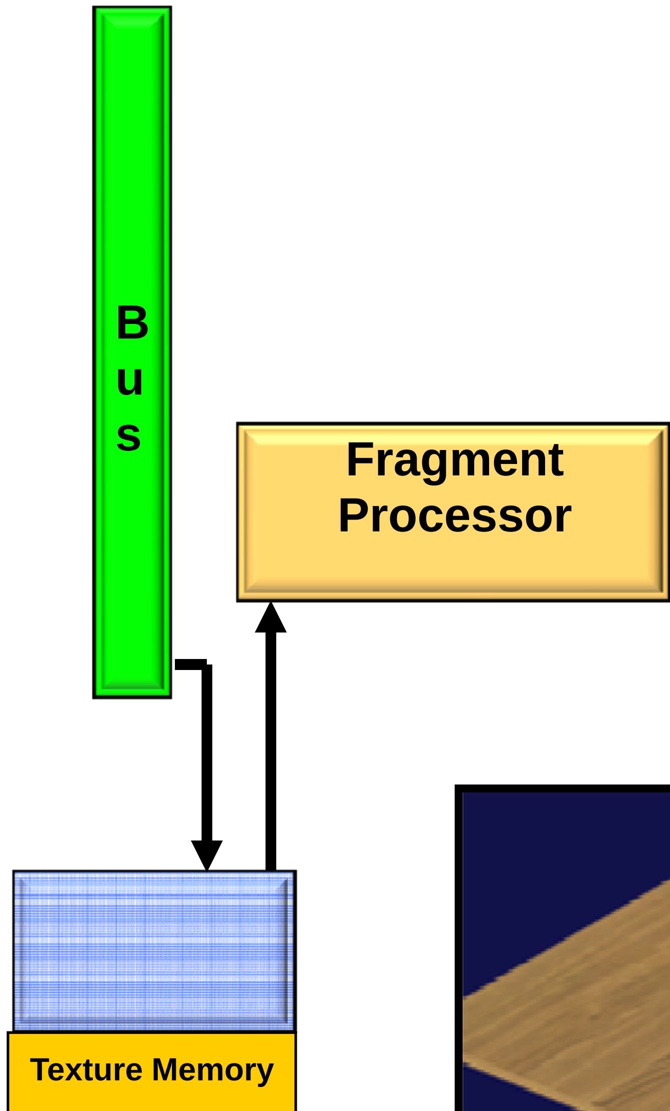
NVIDIA

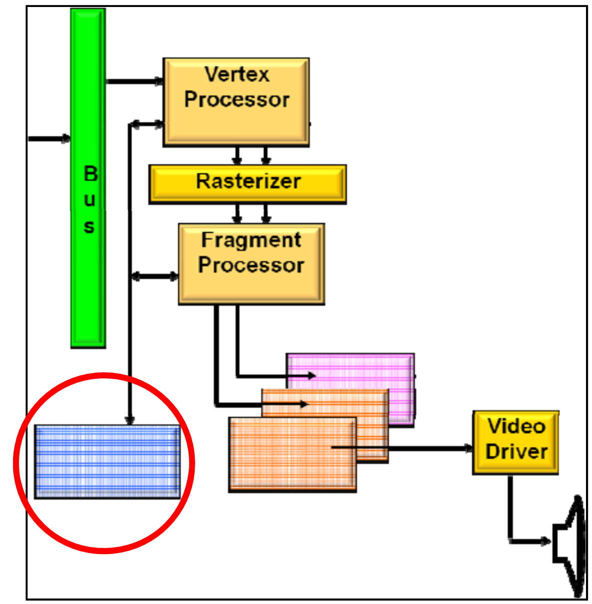
Rasterizers Can Interpolate:

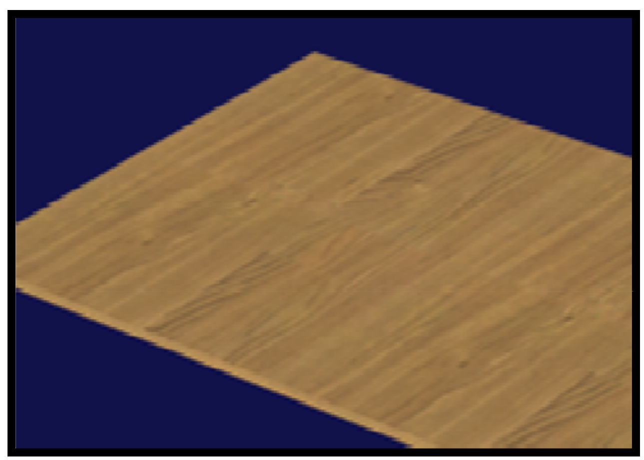
- X and Y
- Red-green-blue values
- Alpha values
- Z values
- Intensities
- Surface normals
- Texture coordinates
- Custom values given by the shaders



Texture Mapping

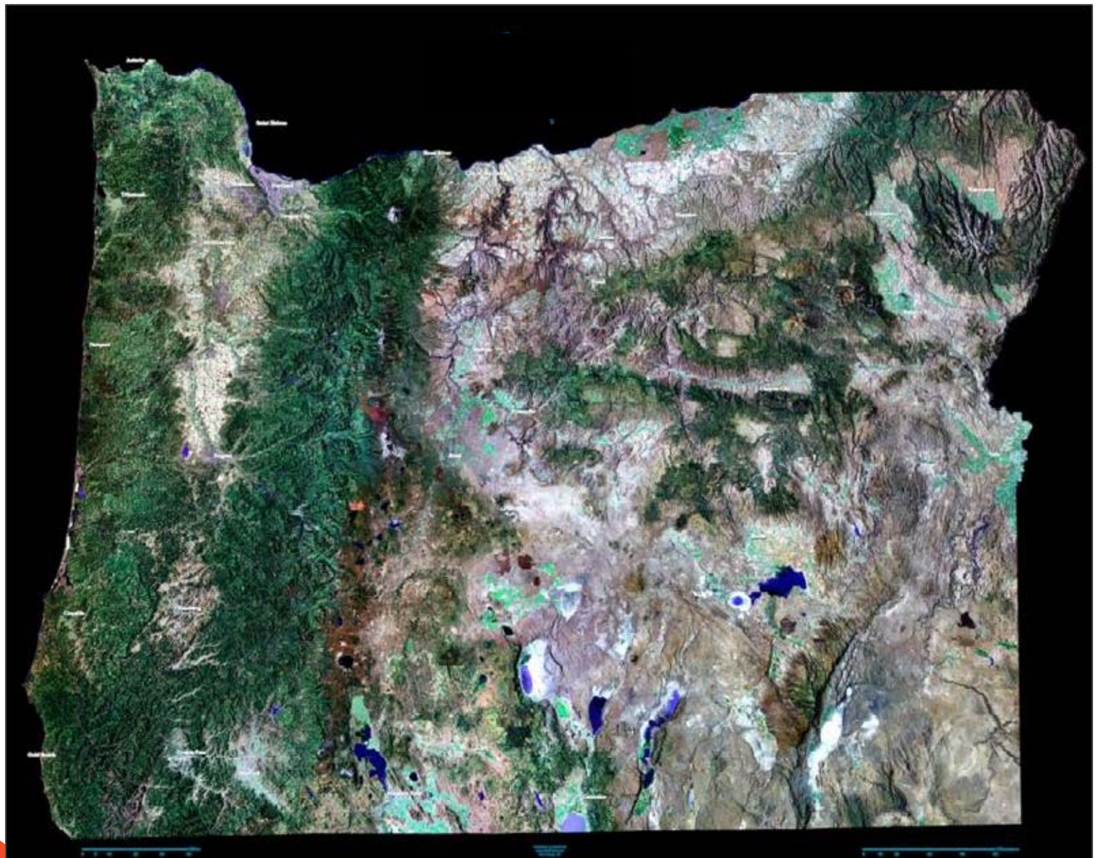


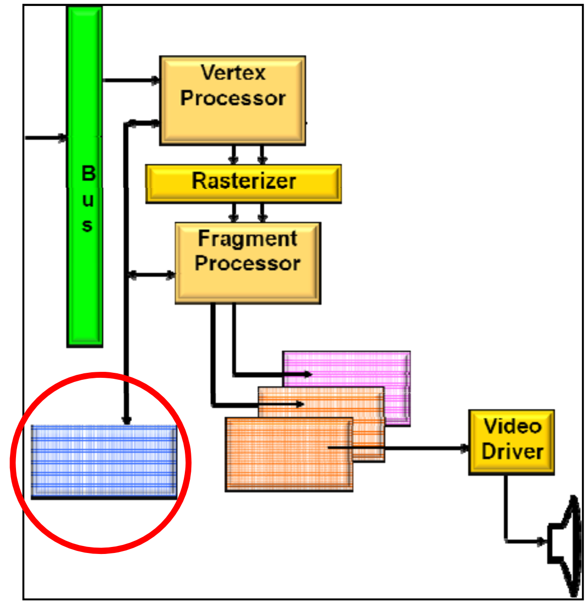




Texture Mapping

- "Stretch" an image onto a piece of geometry
- Image can be generated by a program or scanned in
- Useful for realistic scene generation

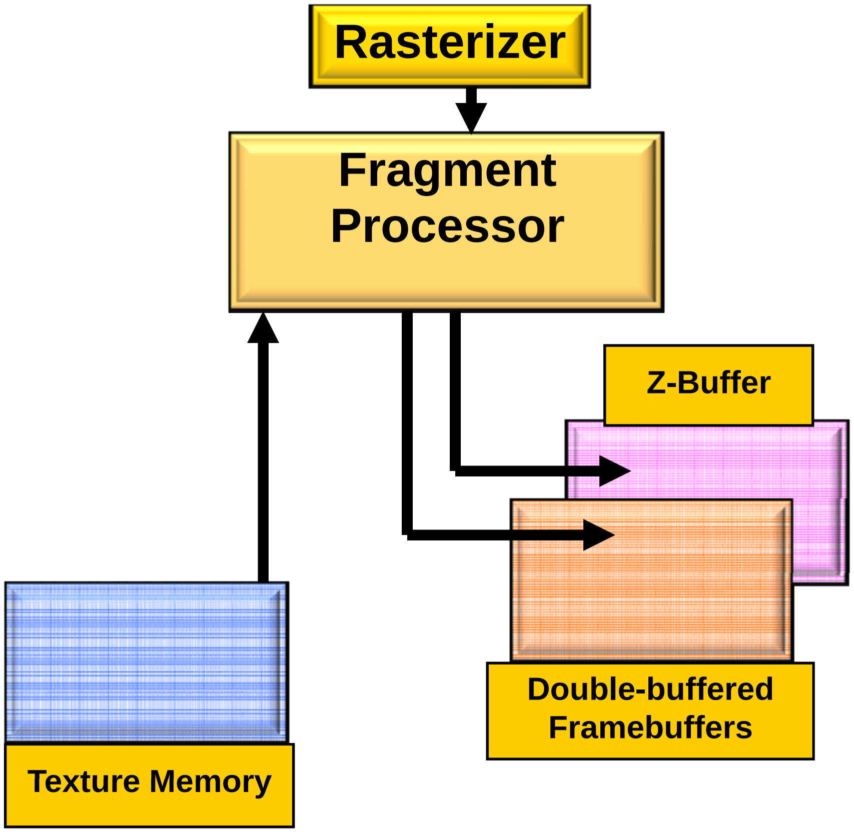


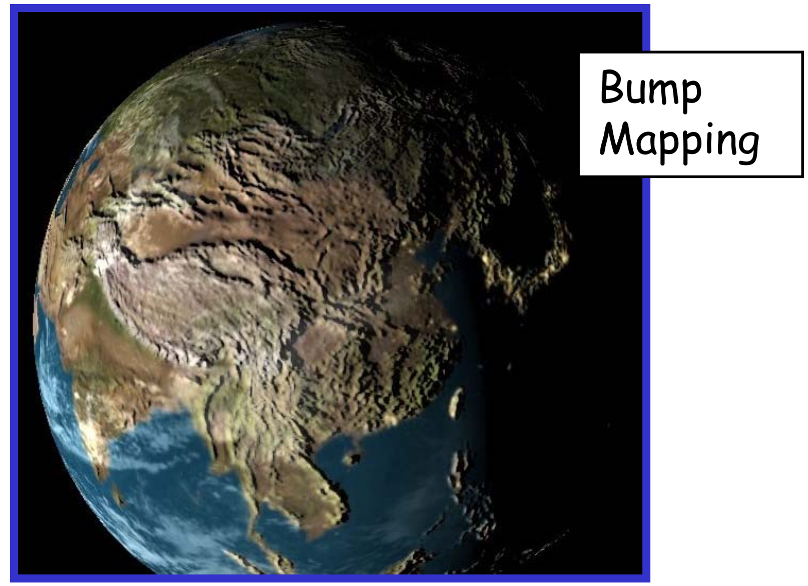


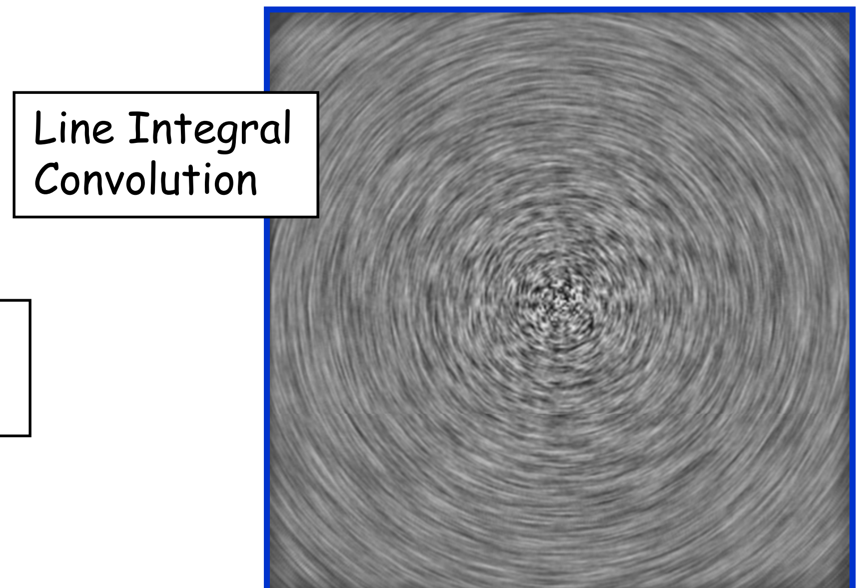
http://2ols.com

Something Cool:

Write-Your-Own Fragment-Processor Code





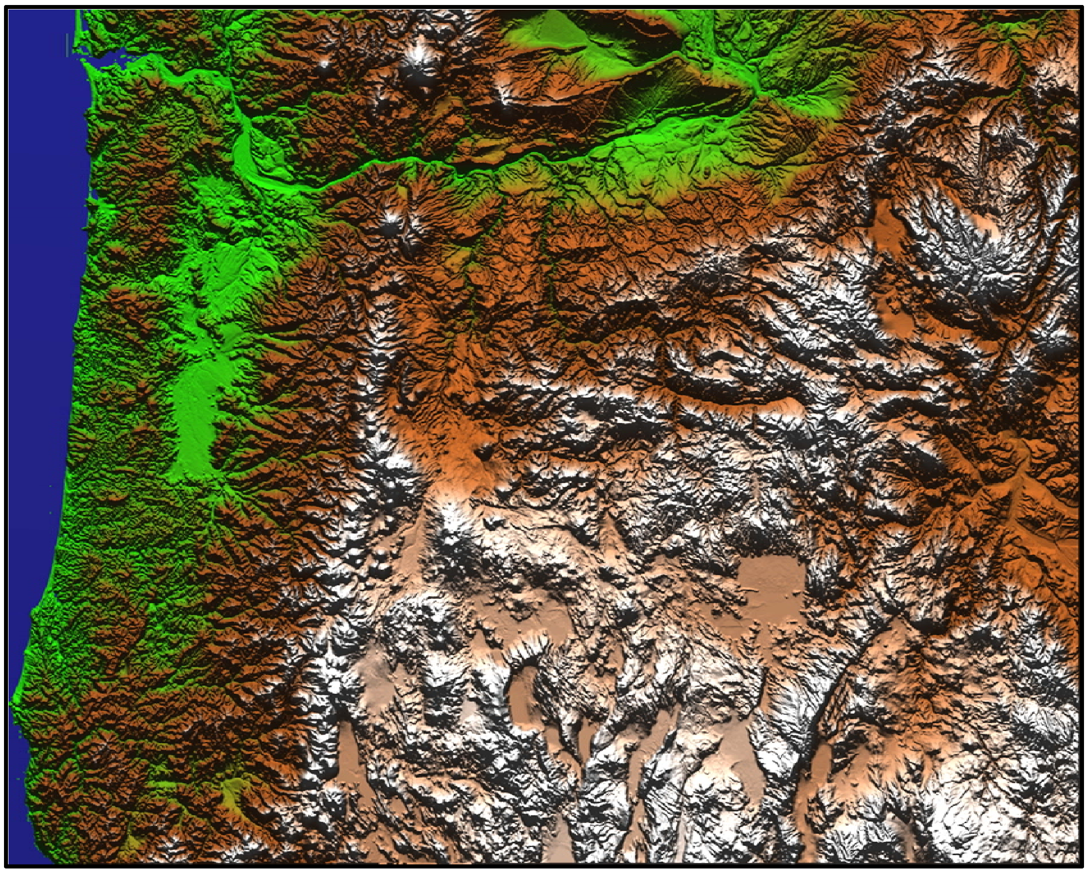


Referred to as:
Pixel Shaders or **Fragment Shaders**

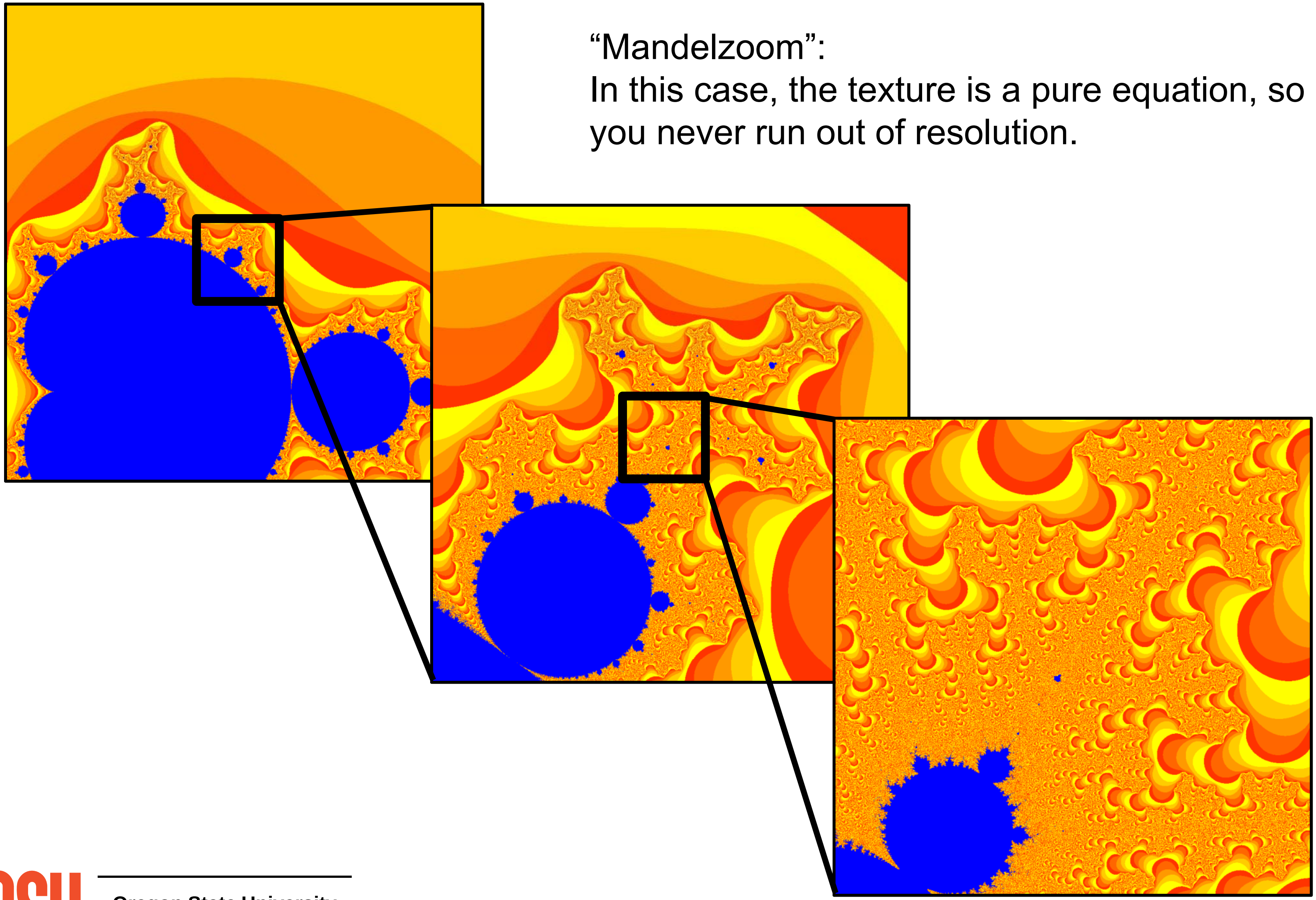
Procedural Texture Mapping

Create a texture from data. In this case, the fragment shader takes a grid of heights and produces surface normals for lighting.

While this is “procedural”, the amount of height data is finite, so you can still run out of resolution



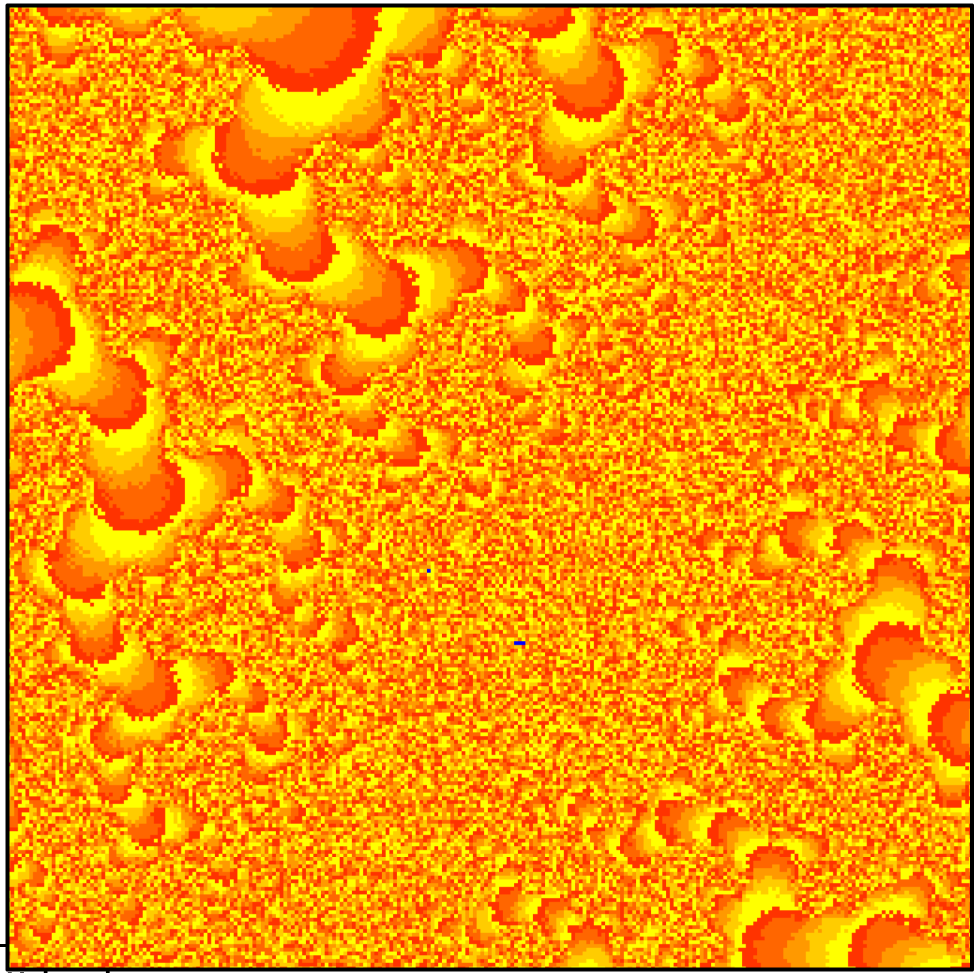
Procedural Texture Mapping



"Mandelzoom":
In this case, the texture is a pure equation, so you never run out of resolution.

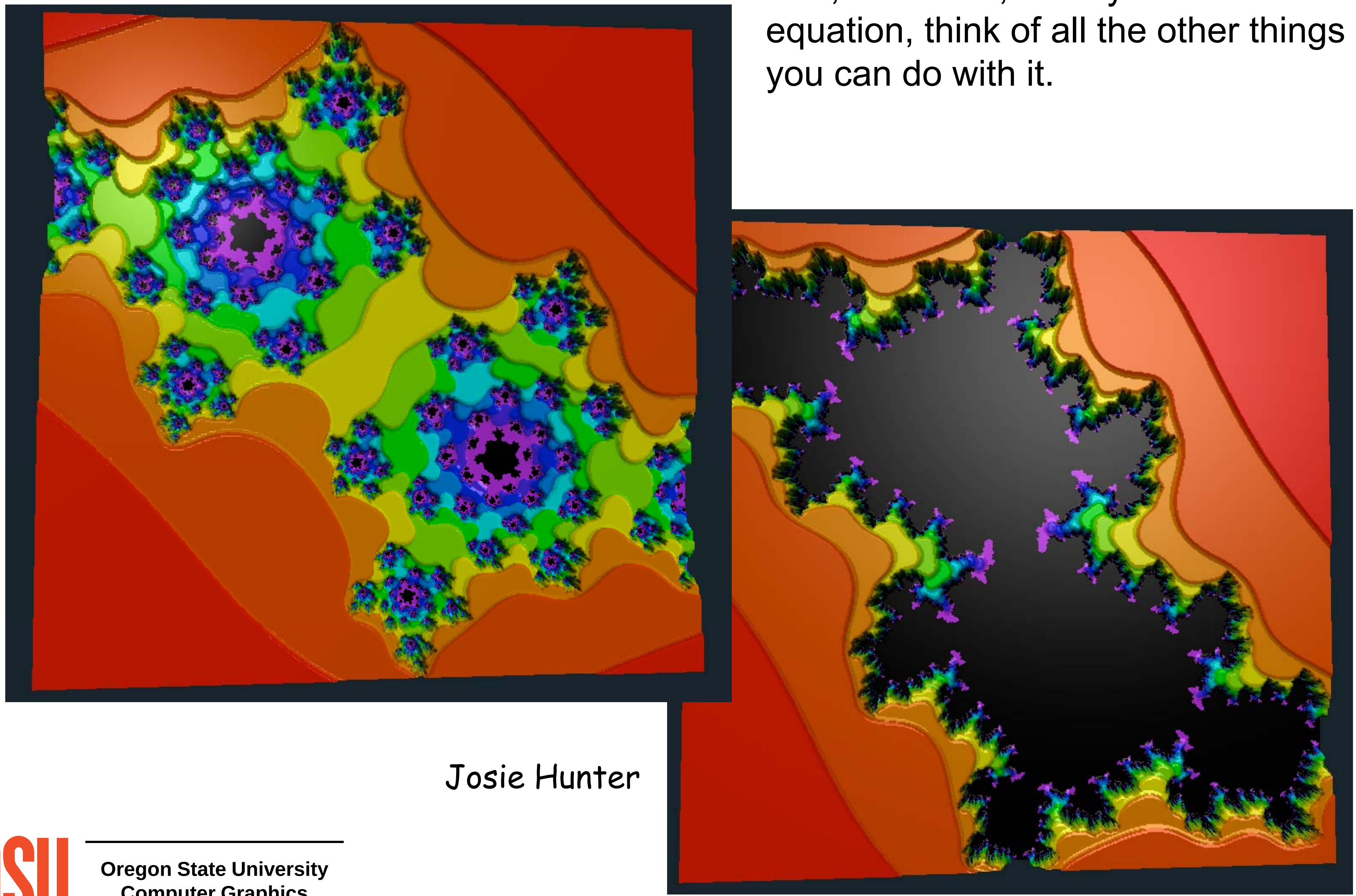
Procedural Texture Mapping

"Mandelzoom":
You can, however, run out of floating point precision:



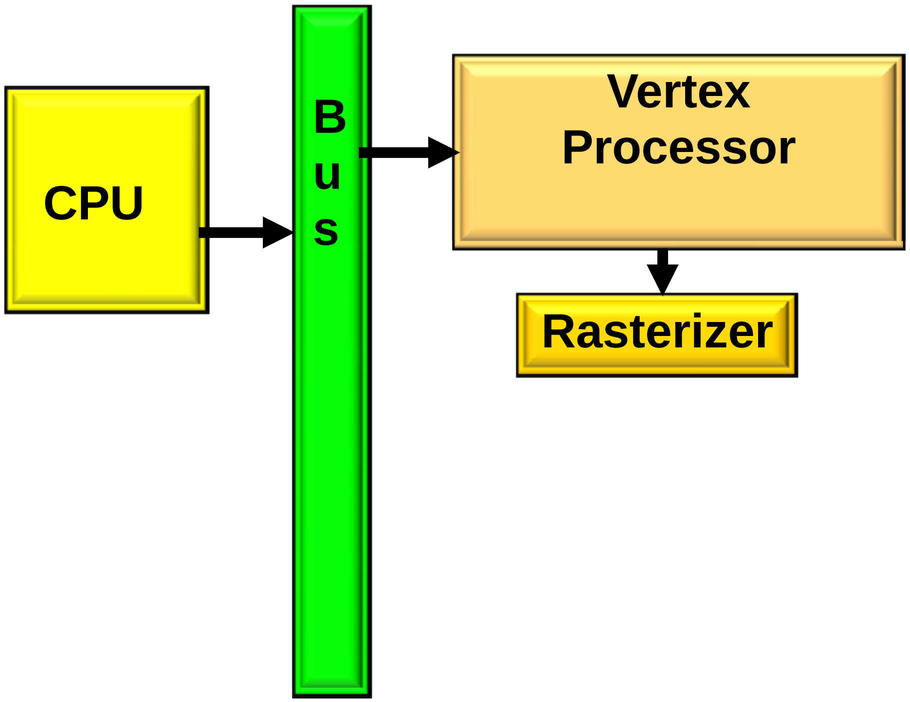
Procedural Texture Mapping

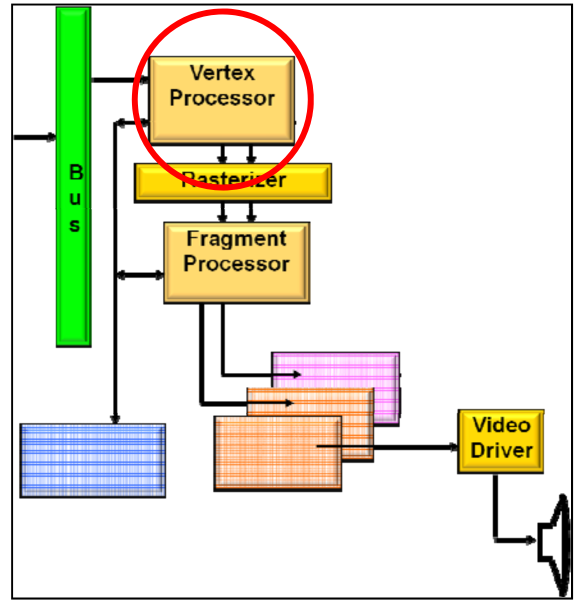
And, of course, once you have an equation, think of all the other things you can do with it.



Josie Hunter

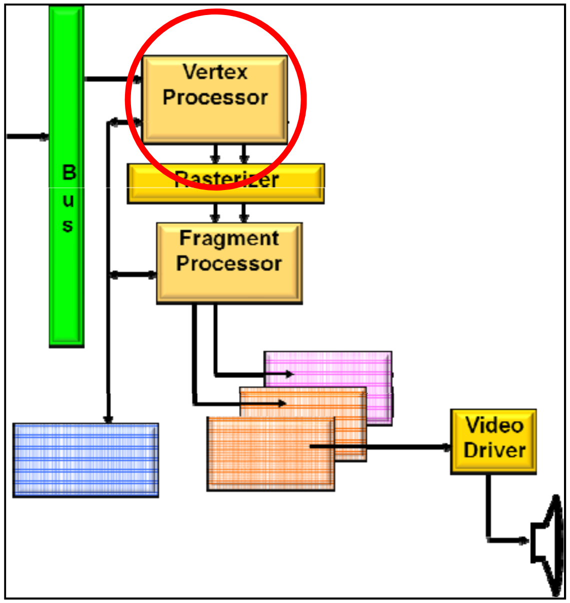
The Vertex Processor





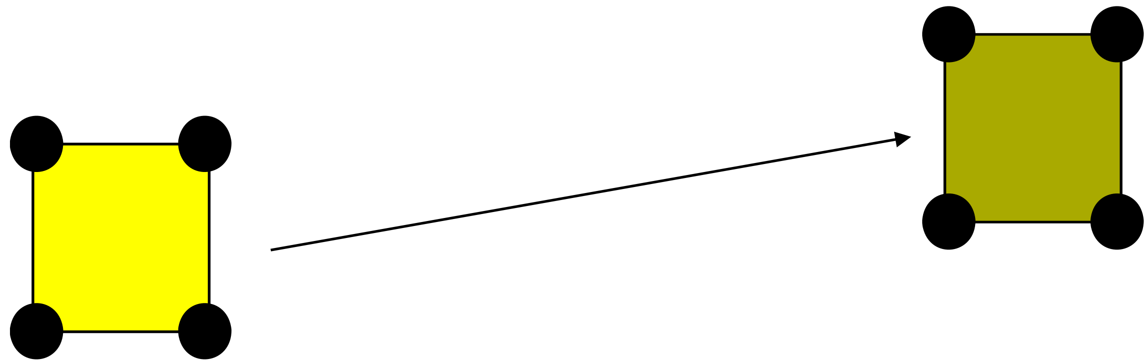
Vertex Processor

- Coordinates enter in model units
- Coordinates leave in screen (pixel) units
- Another great place for custom electronics

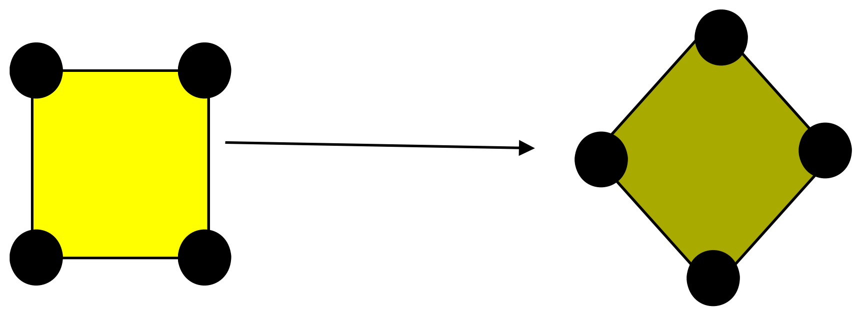


Vertex Processor: Transformations

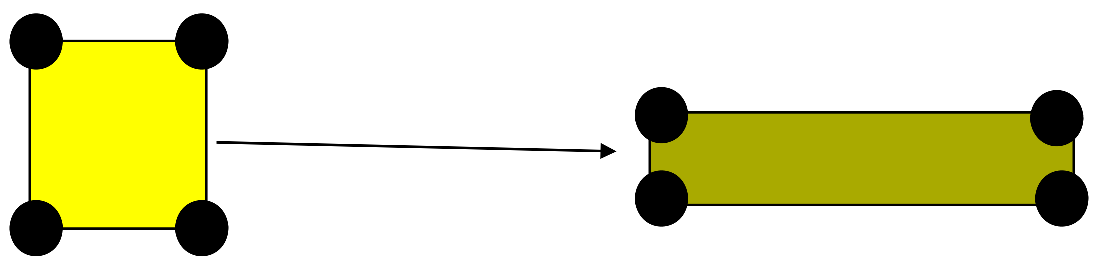
- Used to correctly place objects in the scene
- Translation



- Rotation



- Scaling

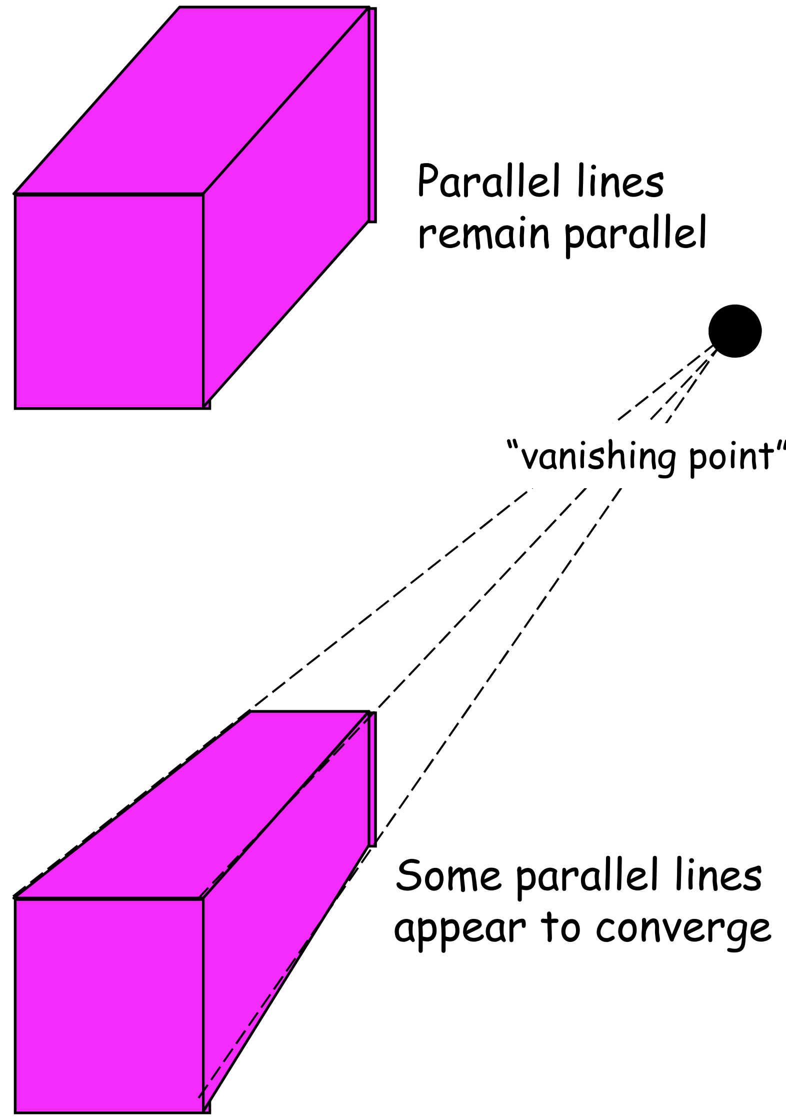


Vertex Processor:
Windowing and Clipping

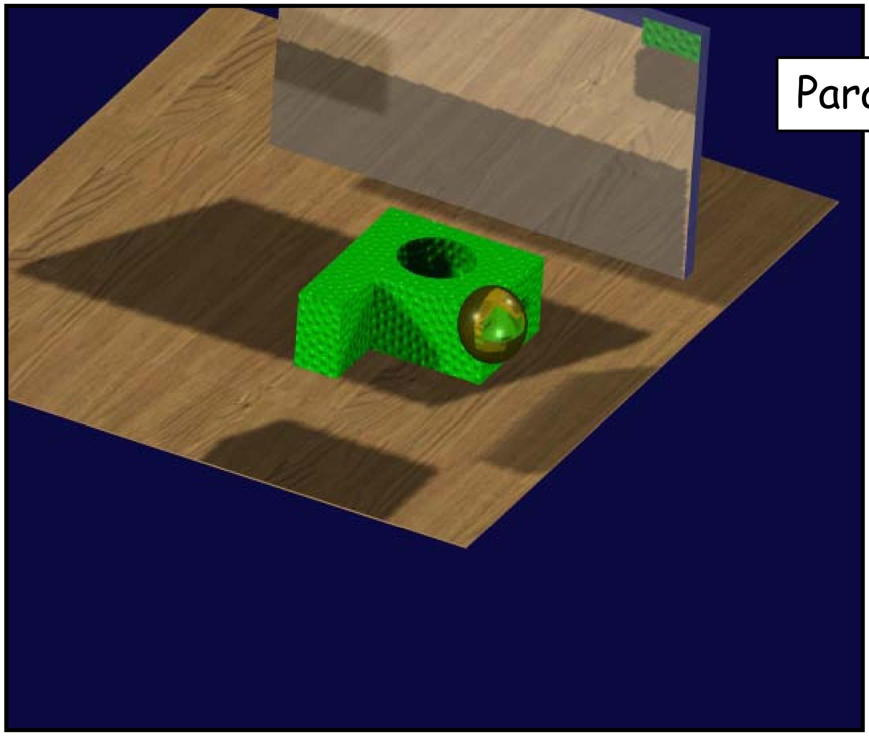
- Declare which portion of the 3D universe you are interested in viewing
- This is called the ***view volume***
- Clip away everything that is outside the viewing volume

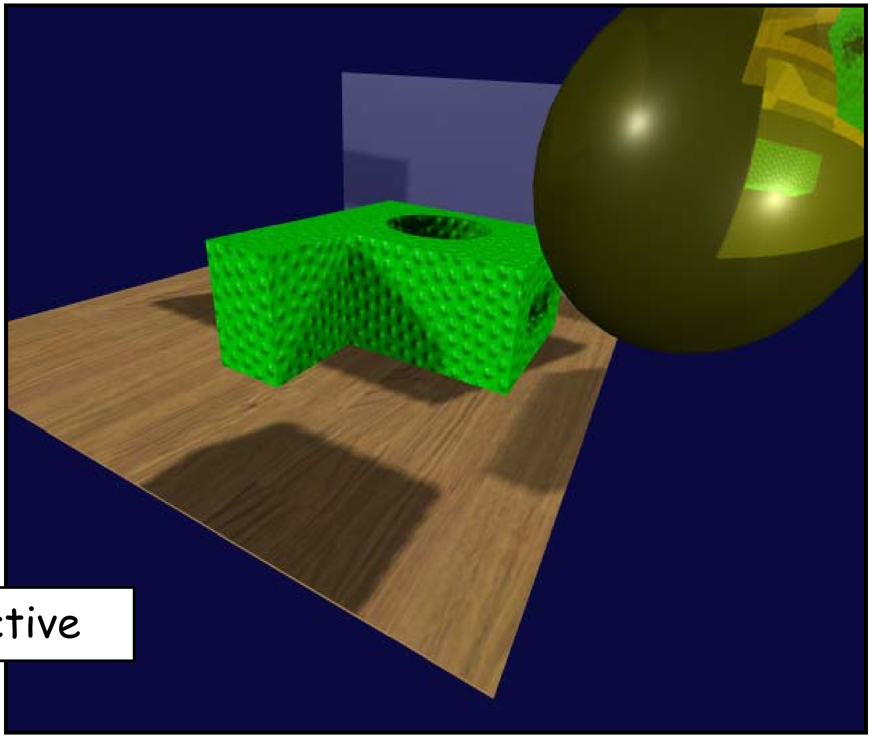
Vertex Processor: Projection

- **Turn 3D coordinates into 2D**
  - ***Parallel projection***
  - ***Perspective projection***

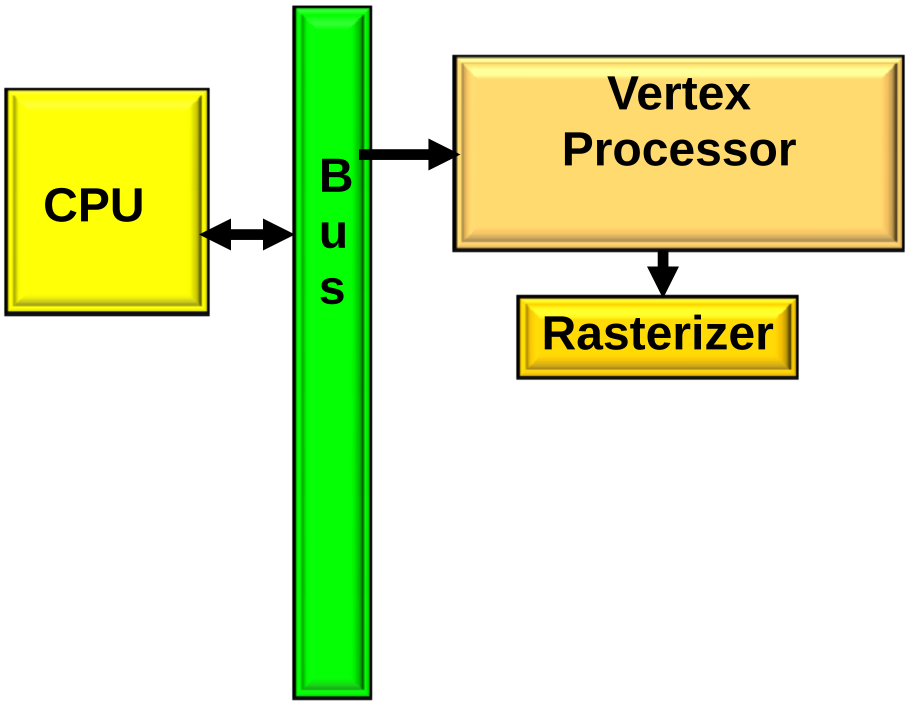


Vertex Processor: Projection

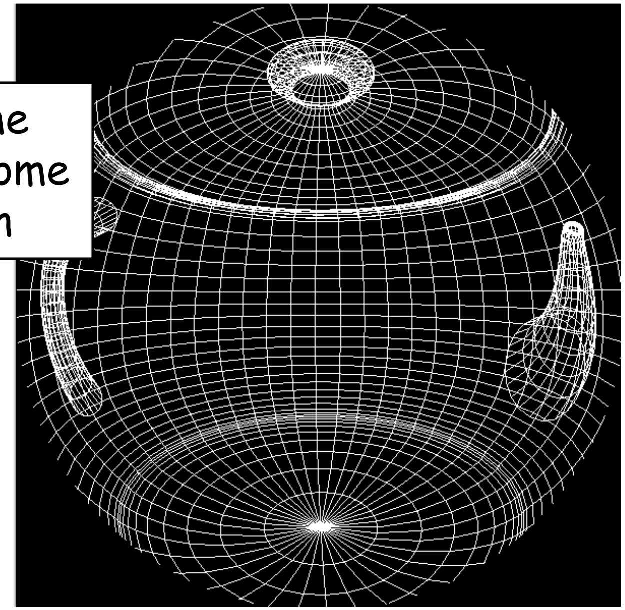




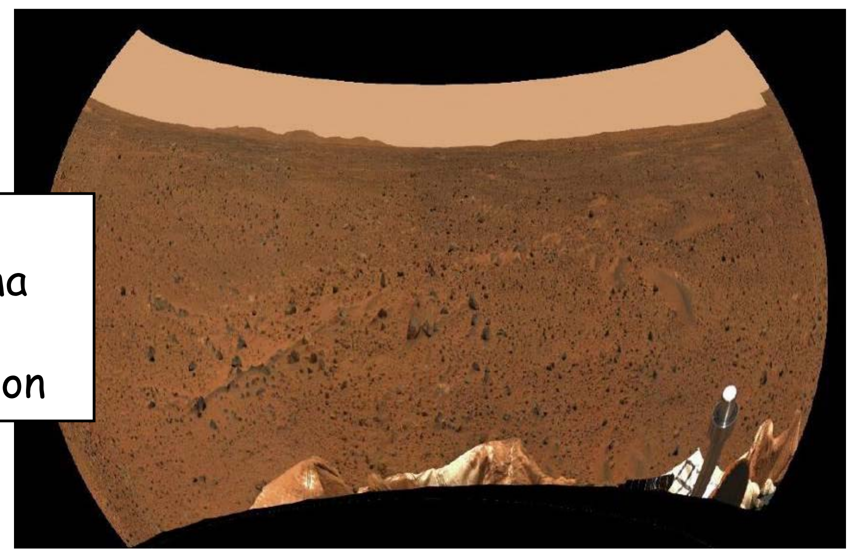
Something Cool:
Write-Your-Own Vertex Code



Referred to as:
Vertex Shaders

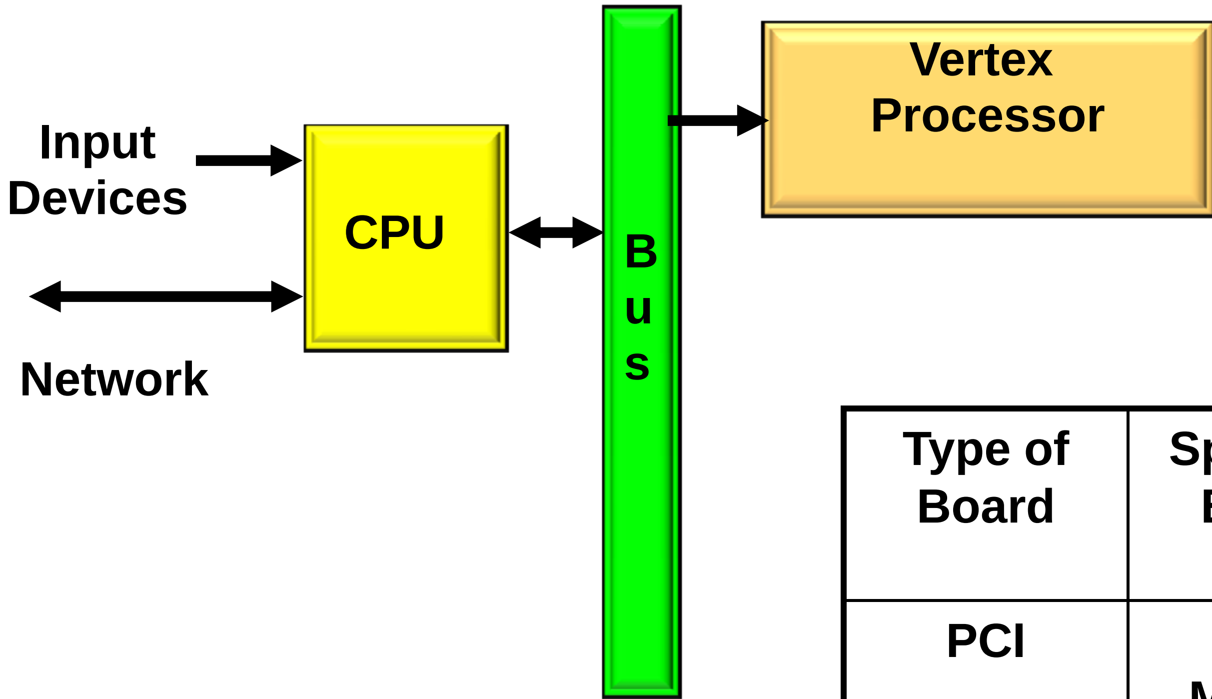


Wireframe Teapot Dome Projection



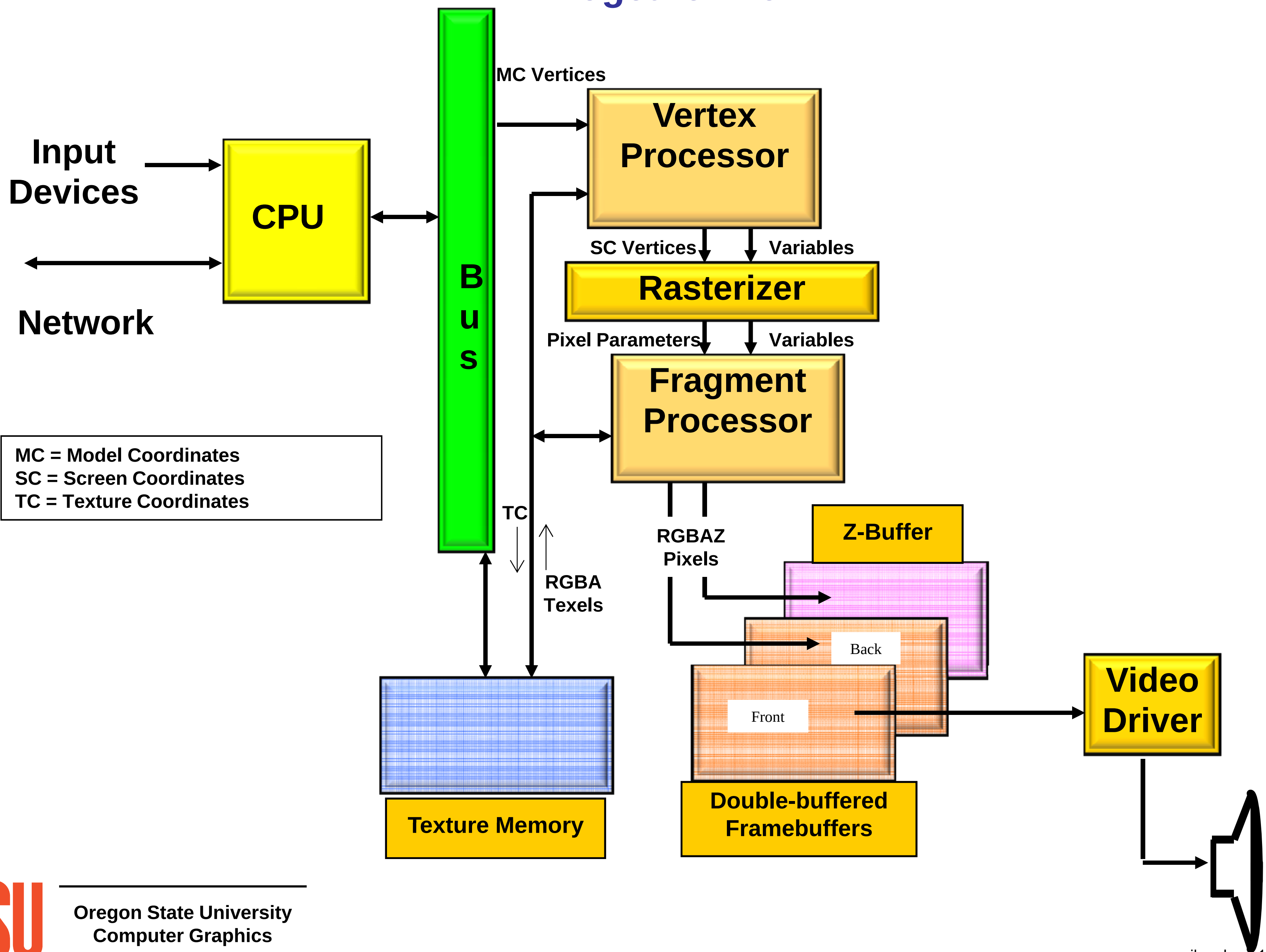
Mars Panorama Dome Projection

The CPU and Bus

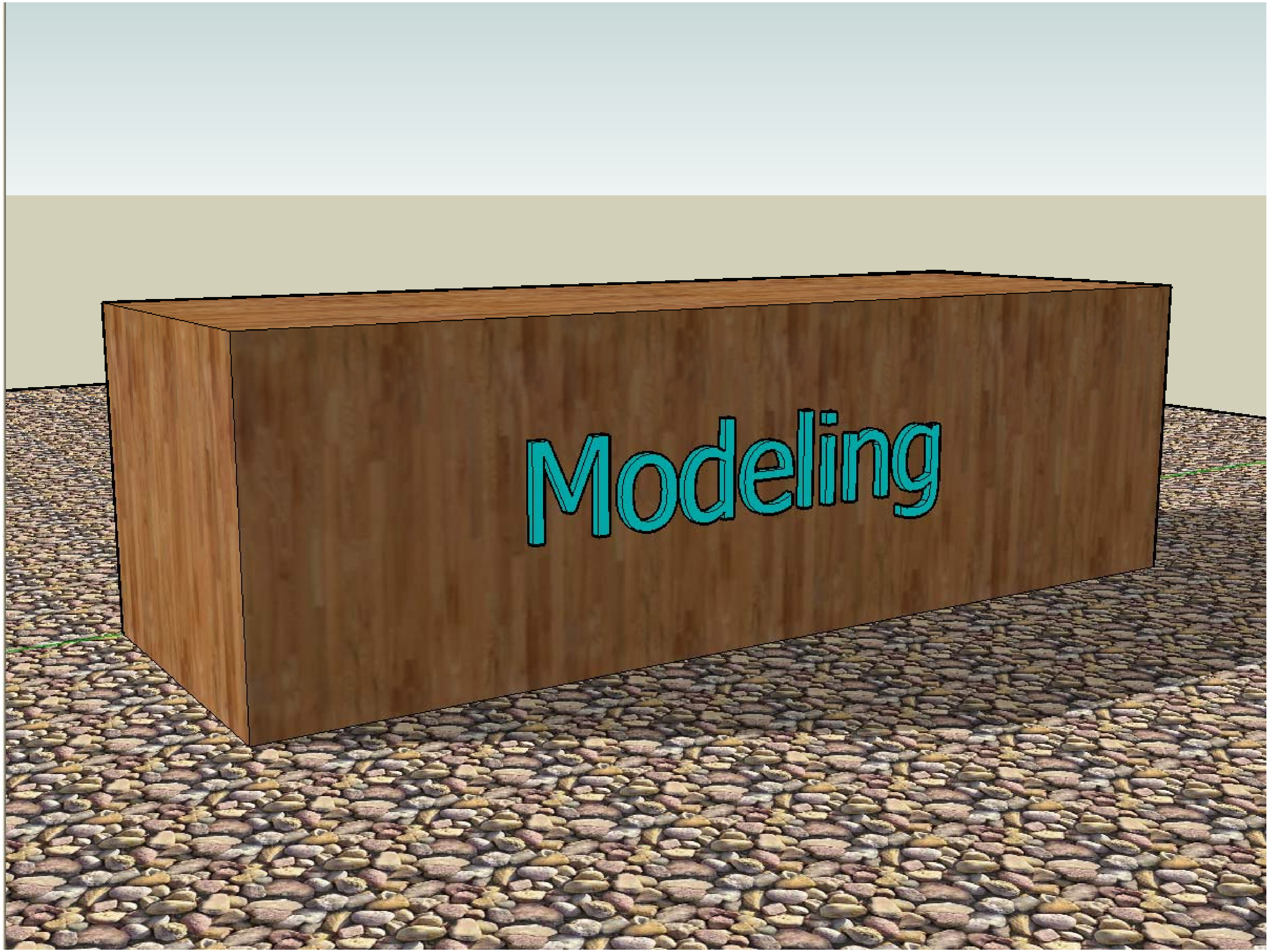


Type of Board	Speed to Board	Speed from Board
PCI	132 Mb/sec	132 Mb/sec
AGP 8X	2 Gb/sec	264 Mb/sec
PCI Express	4 Gb/sec	4 Gb/sec

All Together Now !



MC = Model Coordinates
SC = Screen Coordinates
TC = Texture Coordinates


Modeling

What is a Model?

A is a model of B if A can be used to ask questions about B.

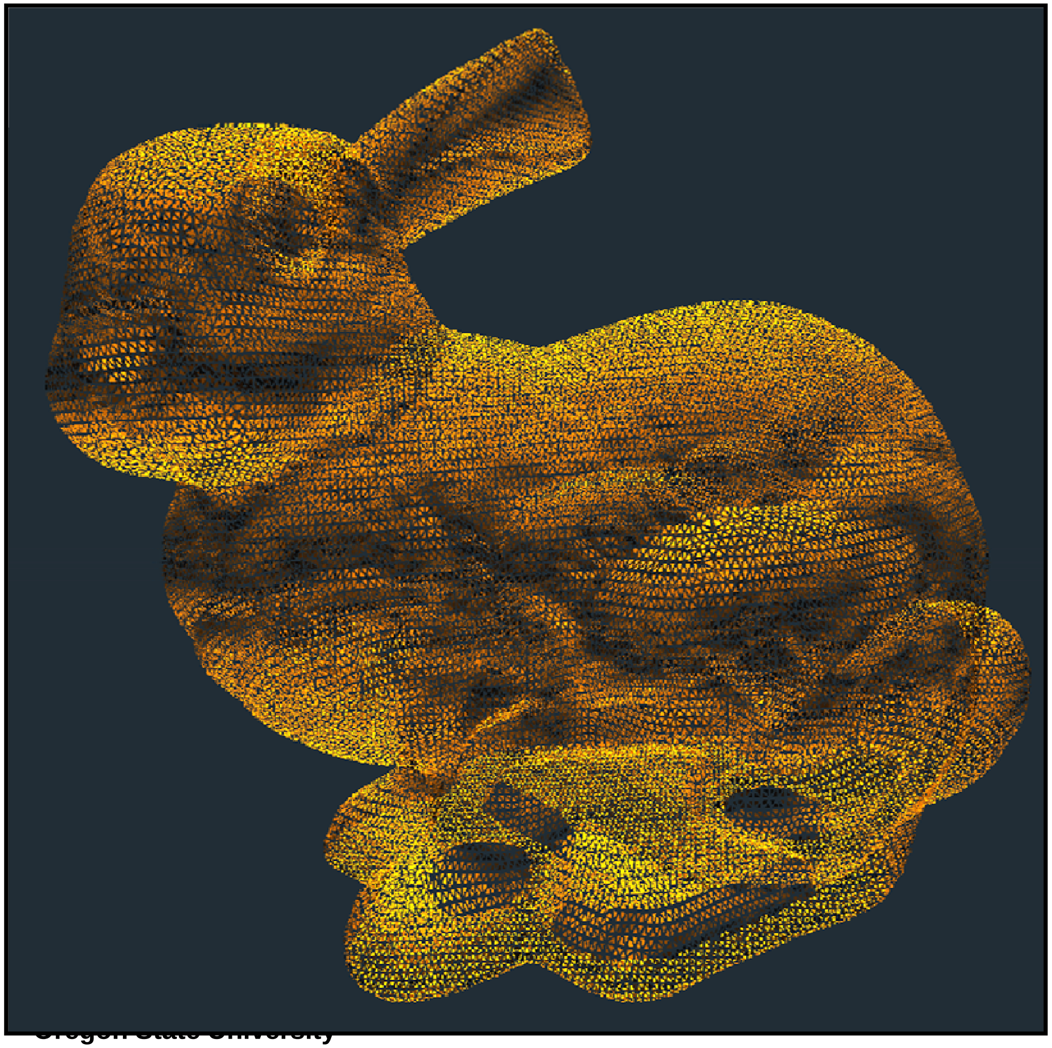
In computer graphics applications, what do we want to ask about B?

- What does B look like?
- How do I want to interact with (shape) B?
- Does B need to be a legal solid?
- How does B interact with its environment?
- What is B's surface area and volume?

These questions, and answers, control what type of geometric modeling you need to do

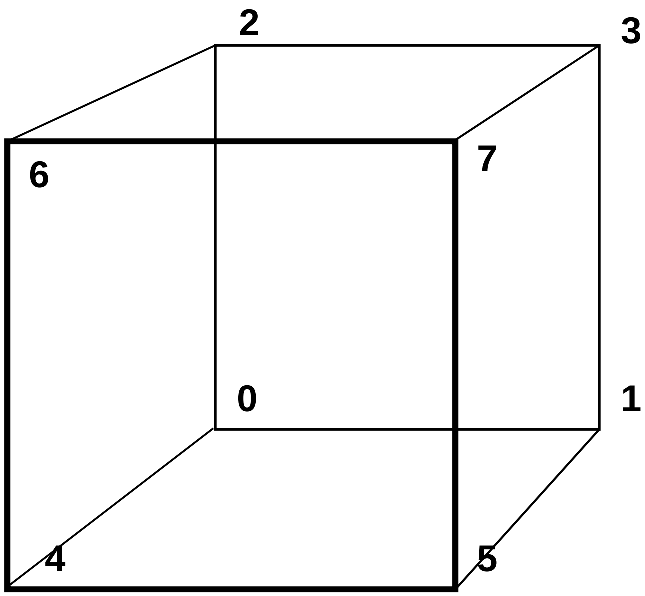
Explicitly Listing Geometry and Topology

Models can consist of thousands of vertices and faces – we need some way to list them efficiently



http://graphics.stanford.edu/data/3Dscanrep

Explicitly Listing Geometry and Topology

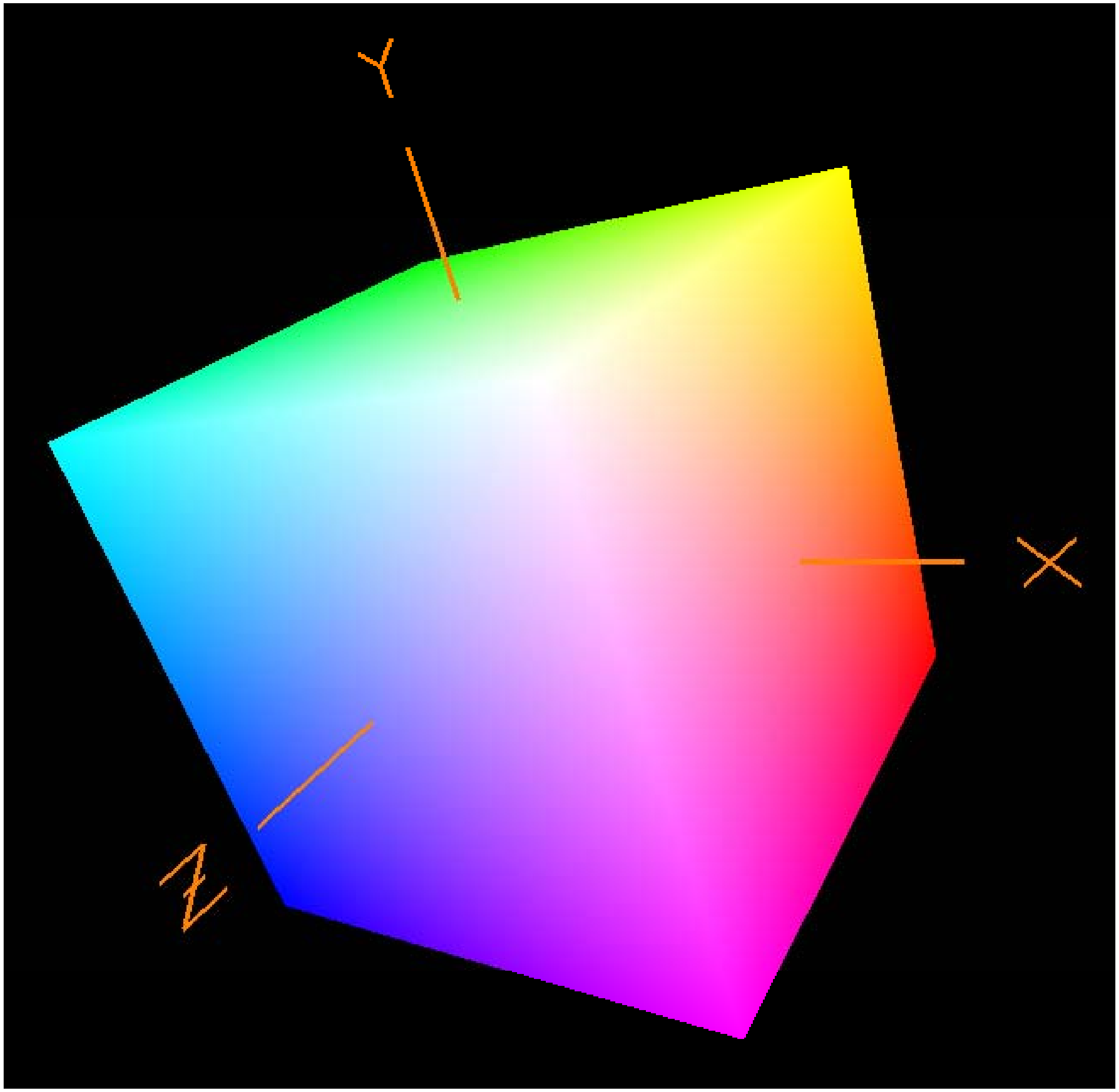



```
static GLfloat CubeVertices[ ][3] =
{
        { -1., -1., -1. },
        {  1., -1., -1. },
        { -1.,  1., -1. },
        {  1.,  1., -1. },
        { -1., -1.,  1. },
        {  1., -1.,  1. },
        { -1.,  1.,  1. },
        {  1.,  1.,  1. }
};
```

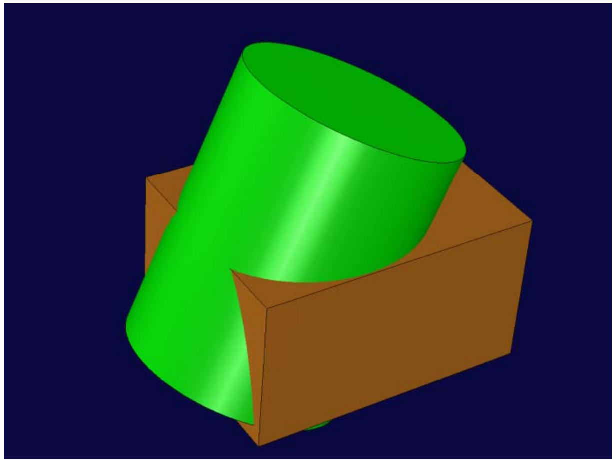
```
static GLfloat CubeColors[ ][3] =
{
        { 0., 0., 0. },
        { 1., 0., 0. },
        { 0., 1., 0. },
        { 1., 1., 0. },
        { 0., 0., 1. },
        { 1., 0., 1. },
        { 0., 1., 1. },
        { 1., 1., 1. },
};
```

```
static GLuint CubeIndices[ ][4] =
{
        { 0, 2, 3, 1 },
        { 4, 5, 7, 6 },
        { 1, 3, 7, 5 },
        { 0, 4, 6, 2 },
        { 2, 6, 7, 3 },
        { 0, 1, 5, 4 }
};
```

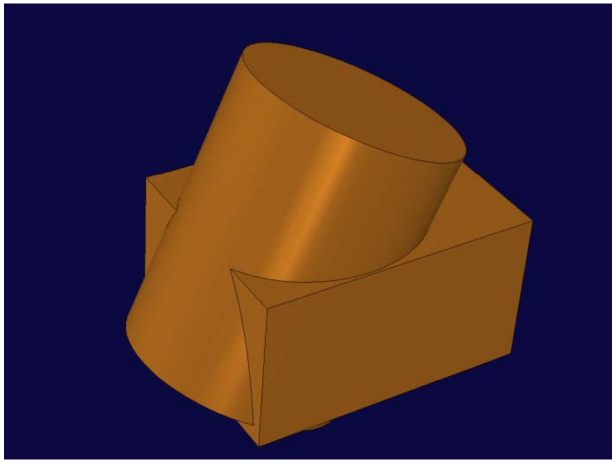
Cube Example



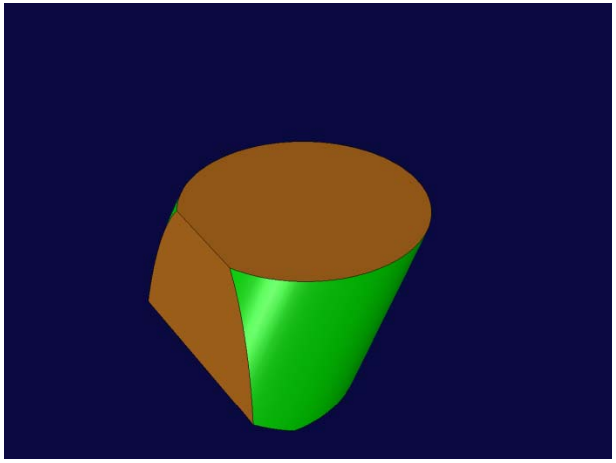
Solid Modeling Using Boolean Operators



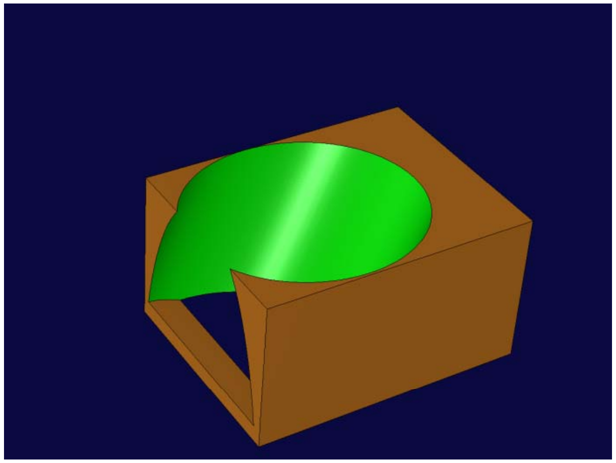
Two Overlapping Solids



Union

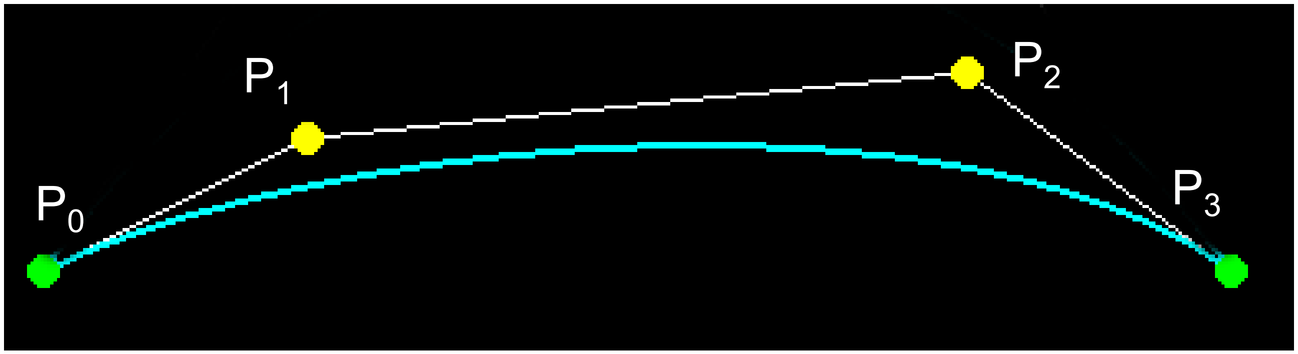


Intersection



Difference

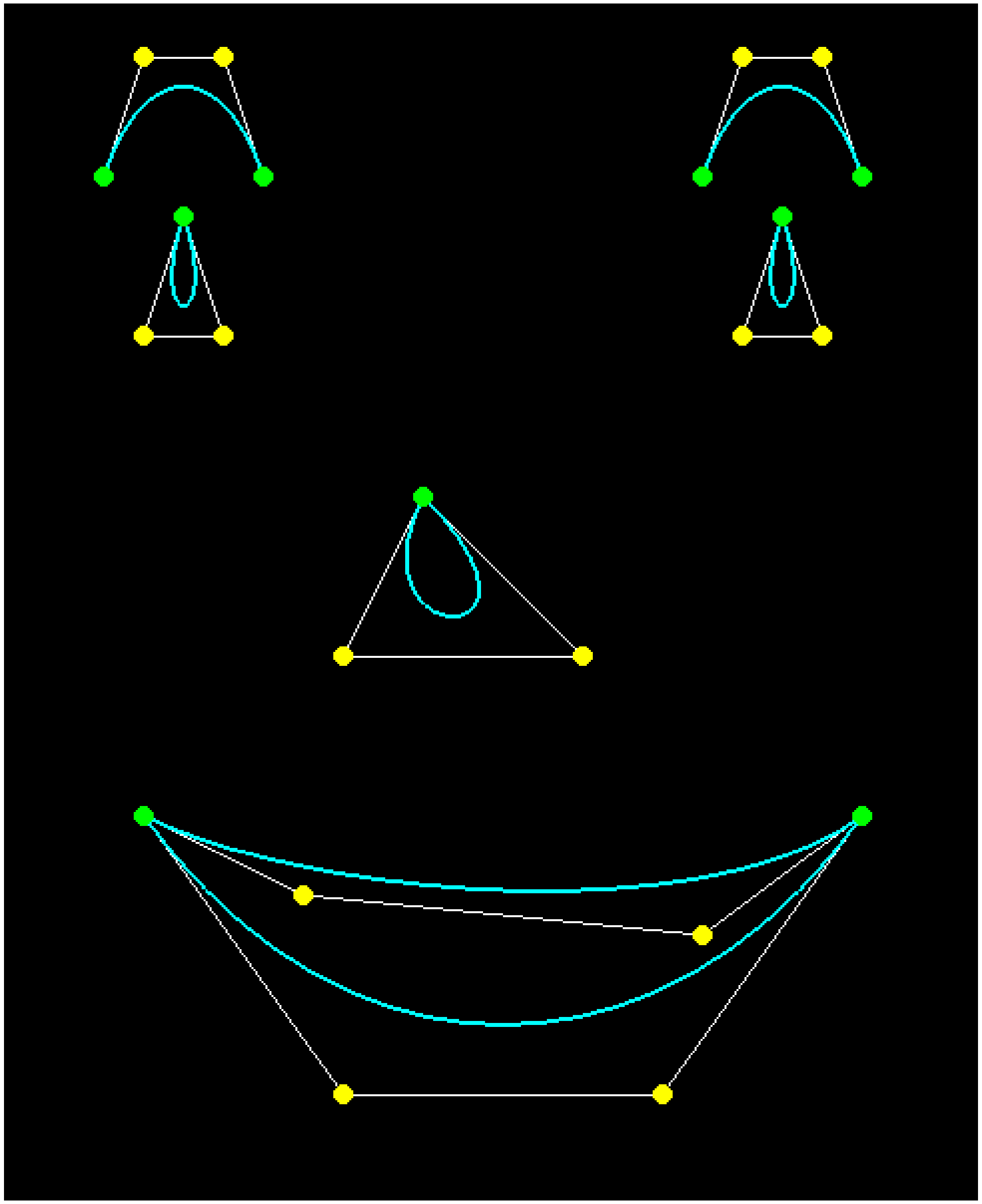
Curve Sculpting – Bezier Curve Sculpting



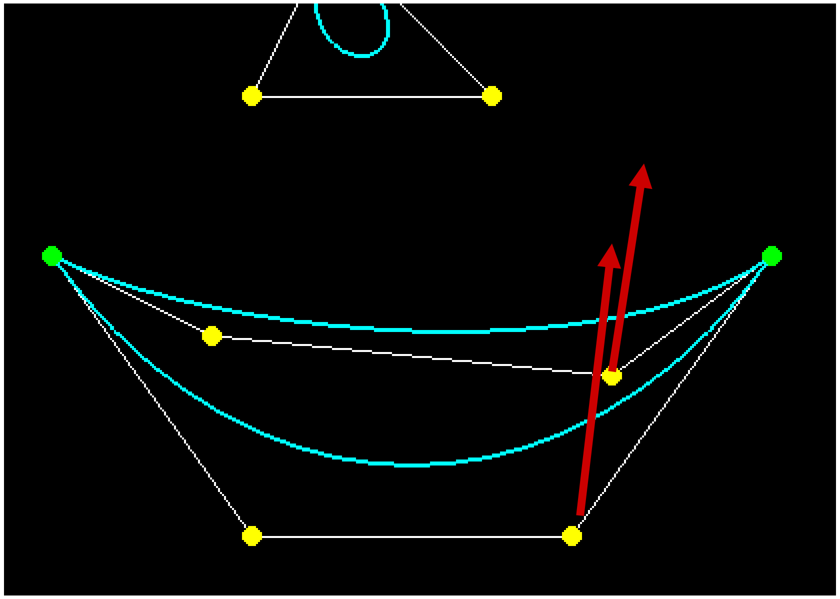
$$P(t) = (1-t)^3 P_0 + 3t(1-t)^2 P_1 + 3t^2(1-t)P_2 + t^3 P_3$$

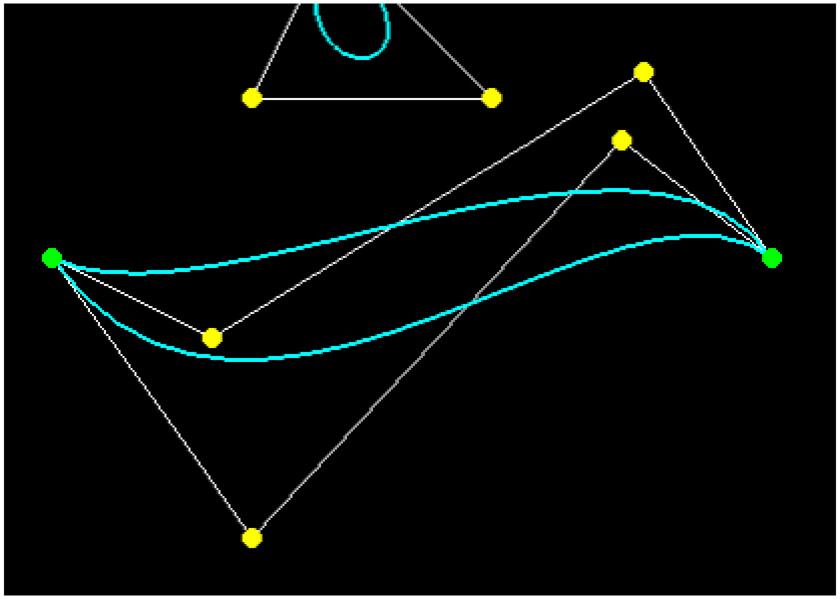
$$0. \le t \le 1.$$

Curve Sculpting – Bezier Curve Sculpting Example

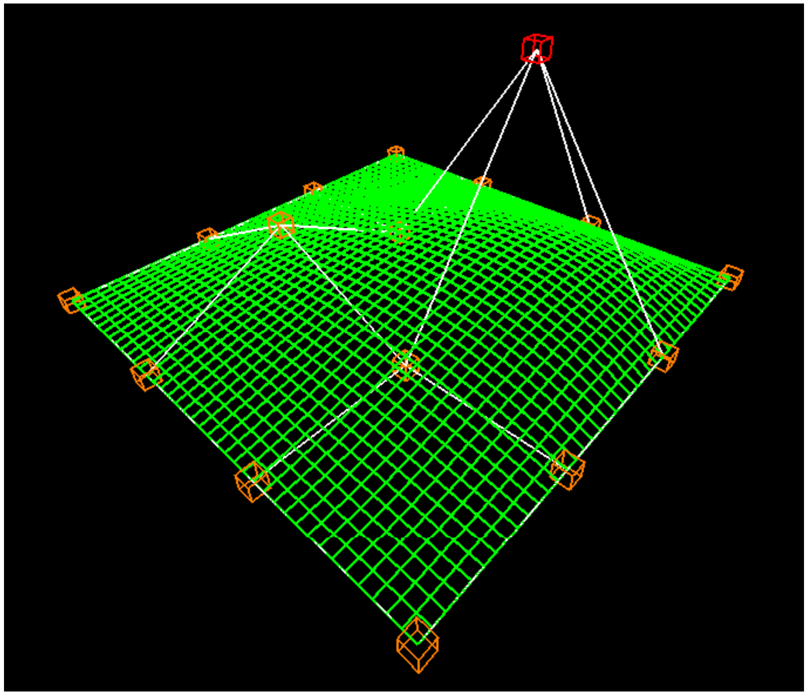


Curve Sculpting – Bezier Curve Sculpting Example

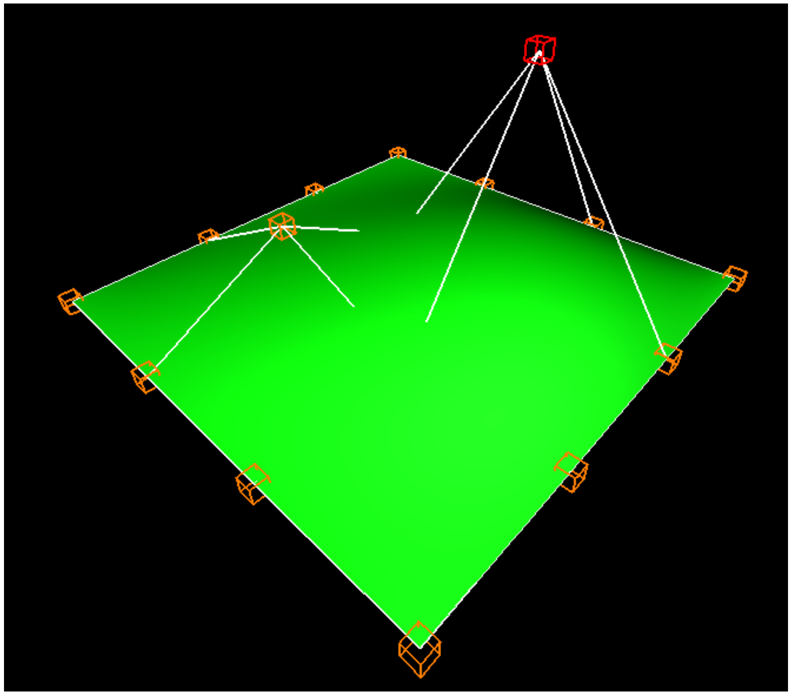




Surface Sculpting

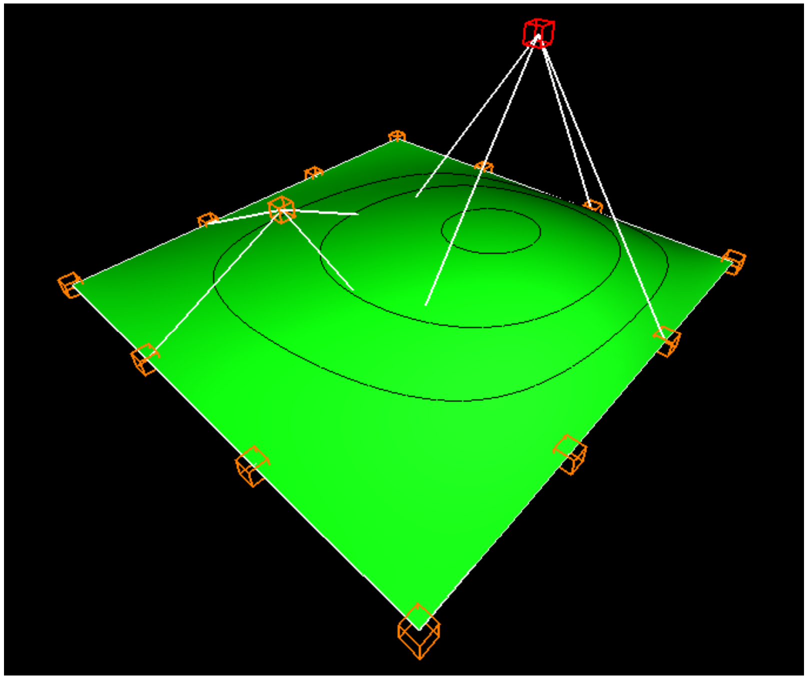


Wireframe

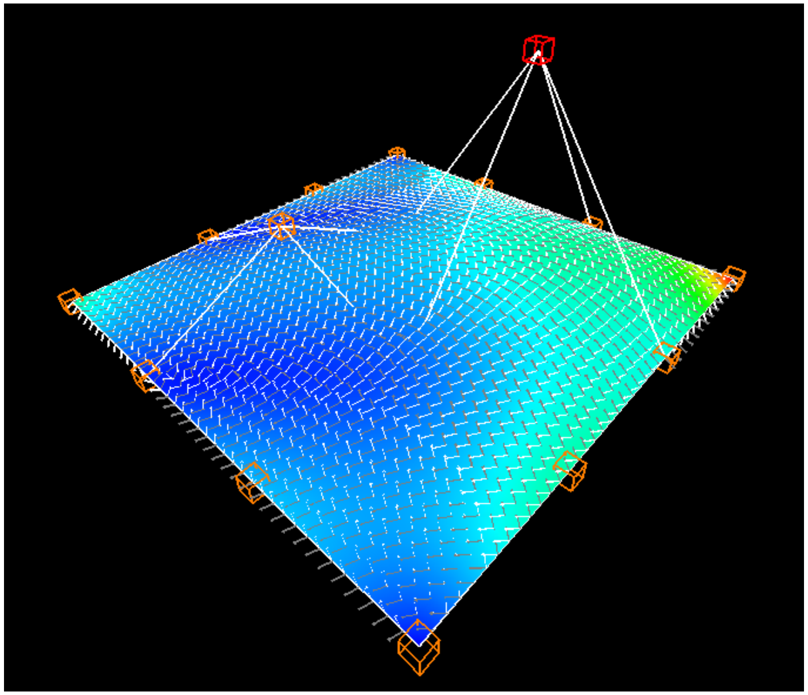


Surface

Surface Equations can also be used for Analysis

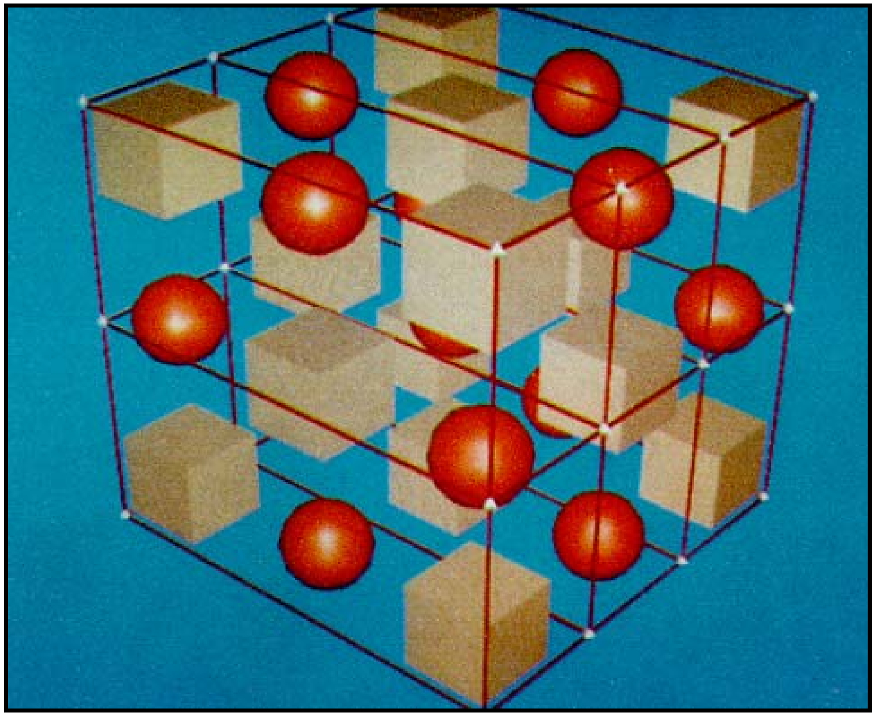


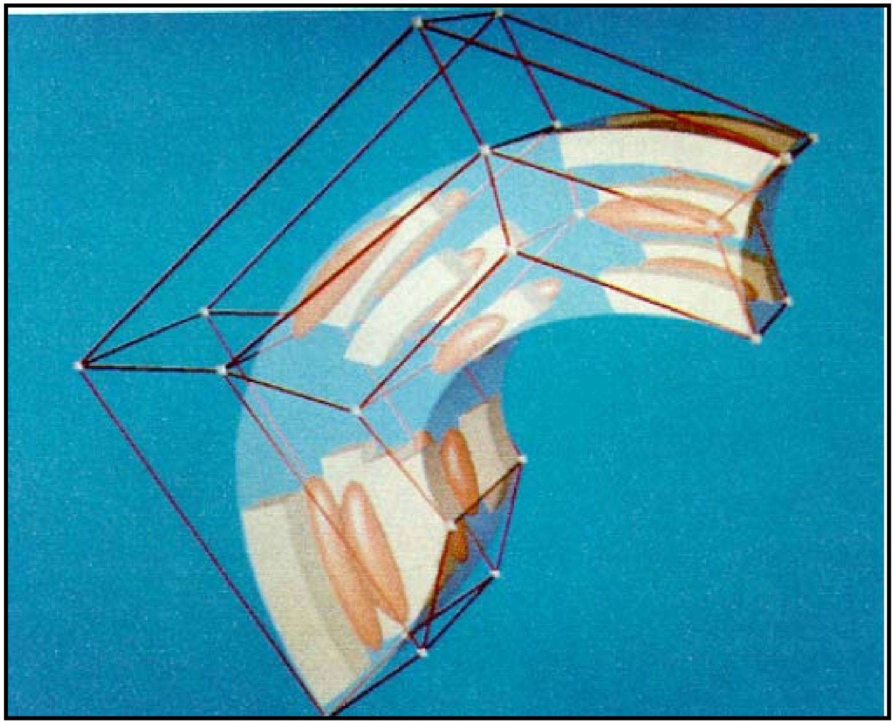
With Contour Lines



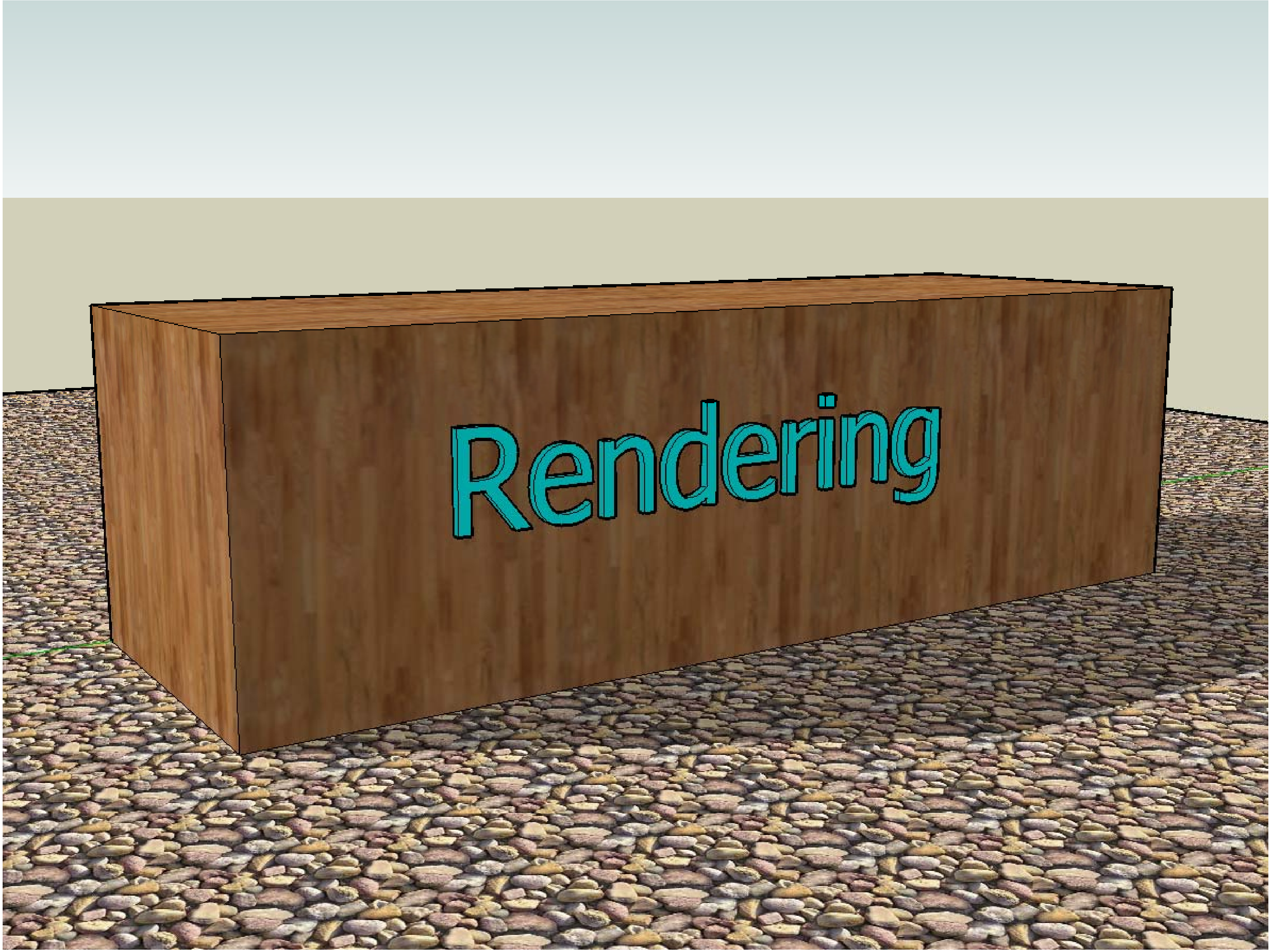
Showing Curvature

Volume Sculpting





Sederberg and Parry

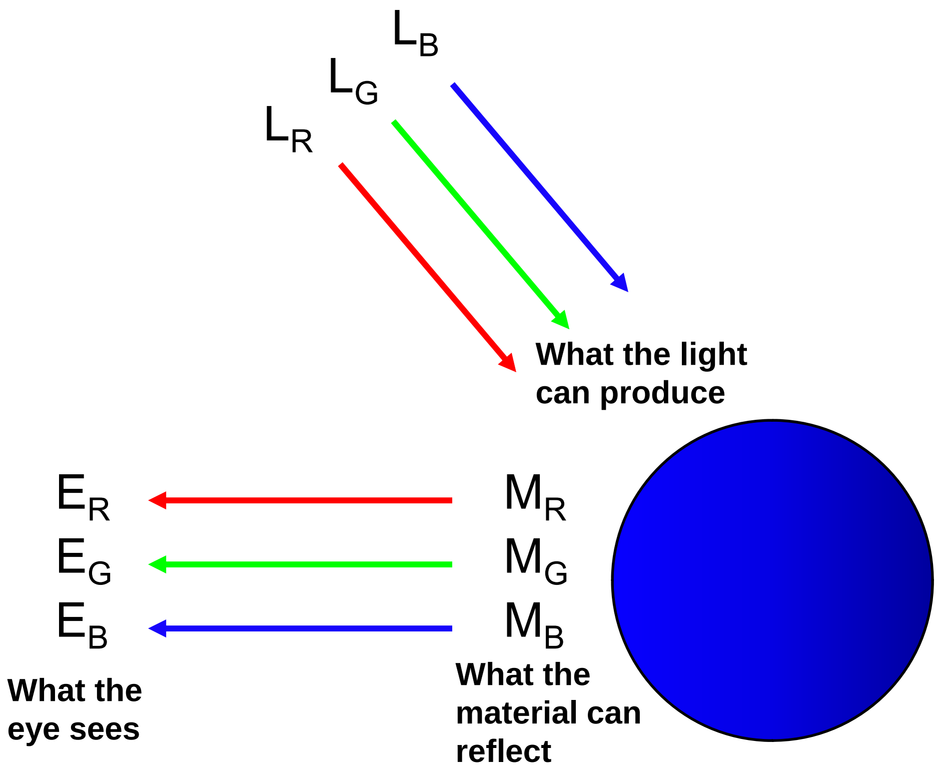

Rendering

Rendering

Rendering is the process of creating an image of a geometric model. Again, there are questions you need to ask:

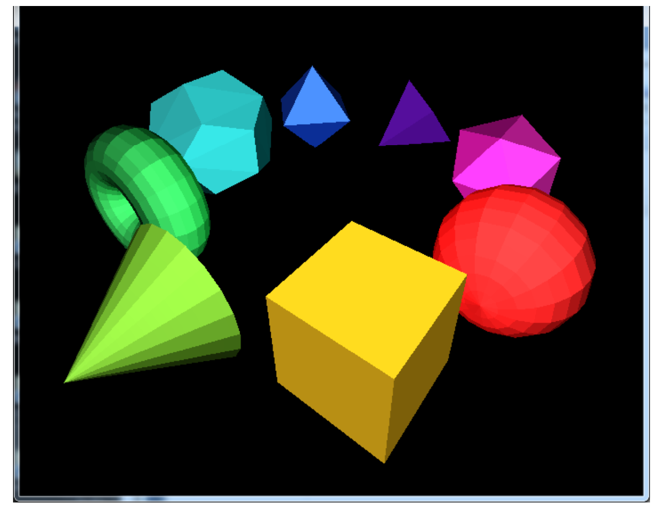
- How realistic do I want this image to be?
- How much compute time do I have to create this image?
- Do I need to take into account lighting?
- Does the illumination need to be global or will local do?
- Do I need to take into account shadows?
- Do I need to take into account reflection and refraction?

Fundamentals of Computer Graphics Lighting

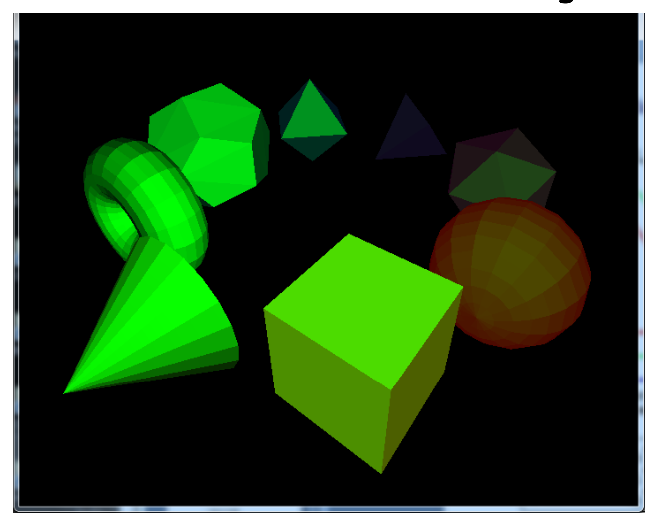


$$E_R = L_R * M_R$$
$$E_G = L_G * M_G$$
$$E_B = L_B * M_B$$

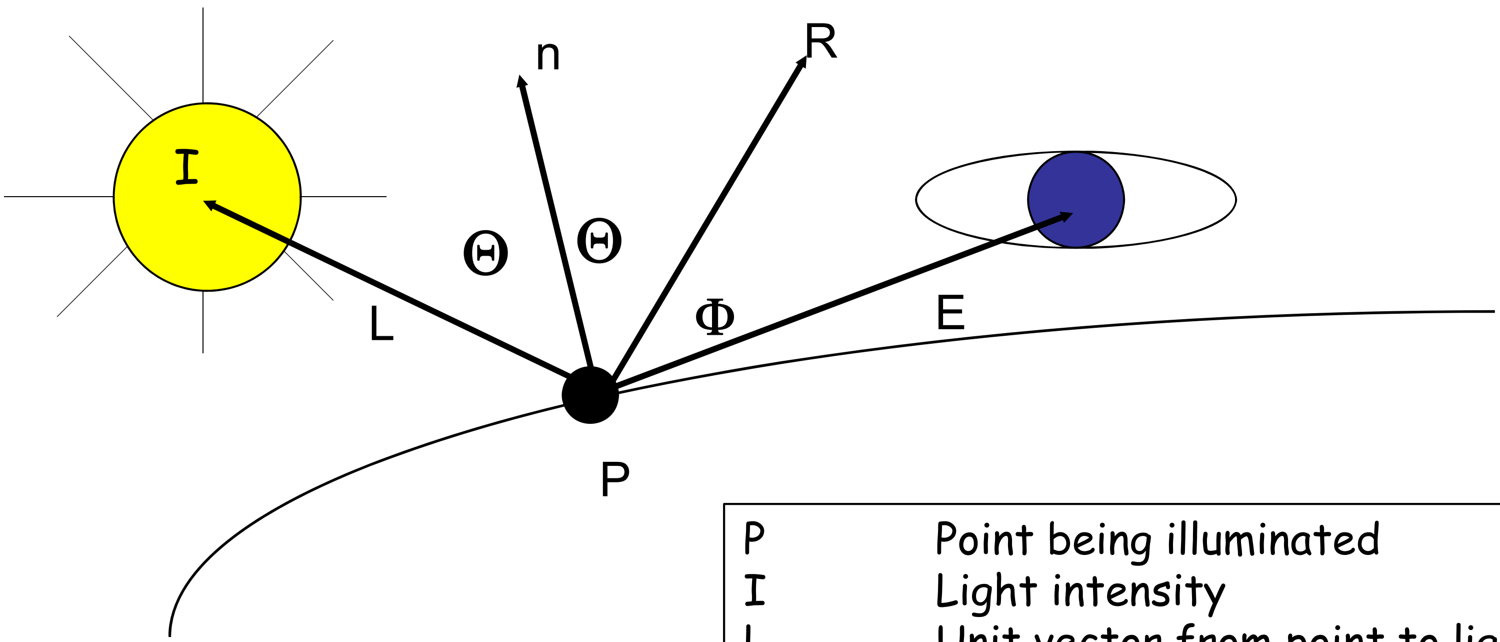
White Light



Green Light



The Computer Graphics Lighting Environment



P	Point being illuminated
I	Light intensity
L	Unit vector from point to light
n	Unit vector surface normal
R	Perfect reflection unit vector
E	Unit vector to eye position

Three Elements of Computer Graphics Lighting

1. Ambient = a constant — *Accounts for light bouncing "everywhere"*

2. Diffuse = $I*\cos\Theta$ — *Accounts for the angle between the incoming light and the surface normal*

3. Specular = $I*\cos^{S}\phi$ — *Accounts for the angle between the "perfect reflector" and the eye; also the exponent, S, accounts for surface shininess*

Note that $\cos\Theta$ is just the dot product between unit vectors L and n

Note that $\cos\phi$ is just the dot product between unit vectors R and E

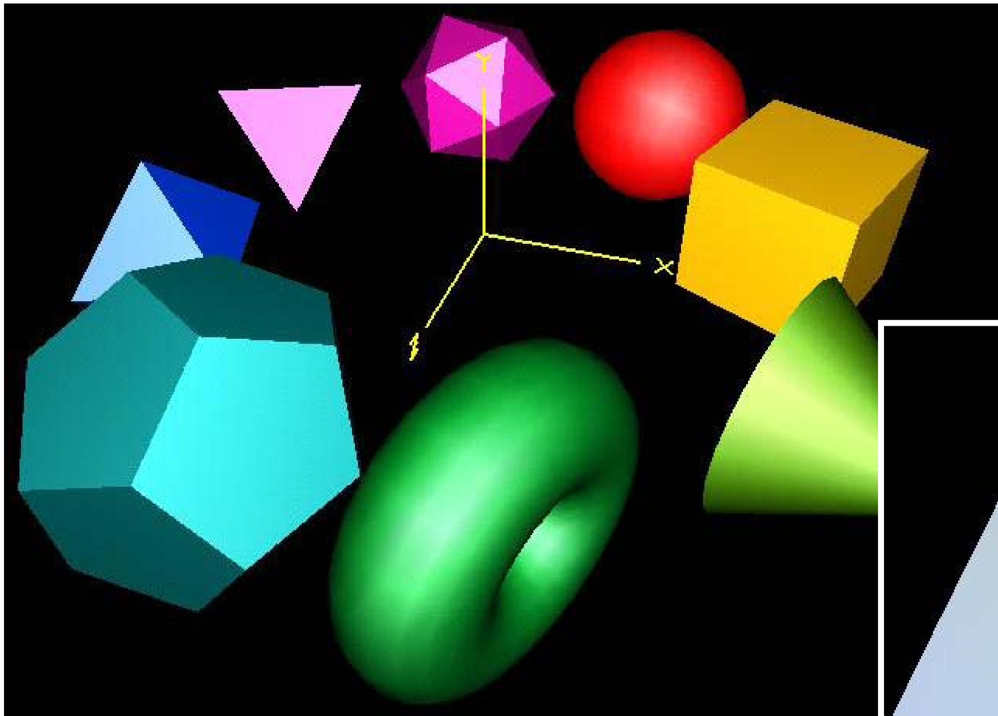
Three Elements of Computer Graphics Lighting

+

+

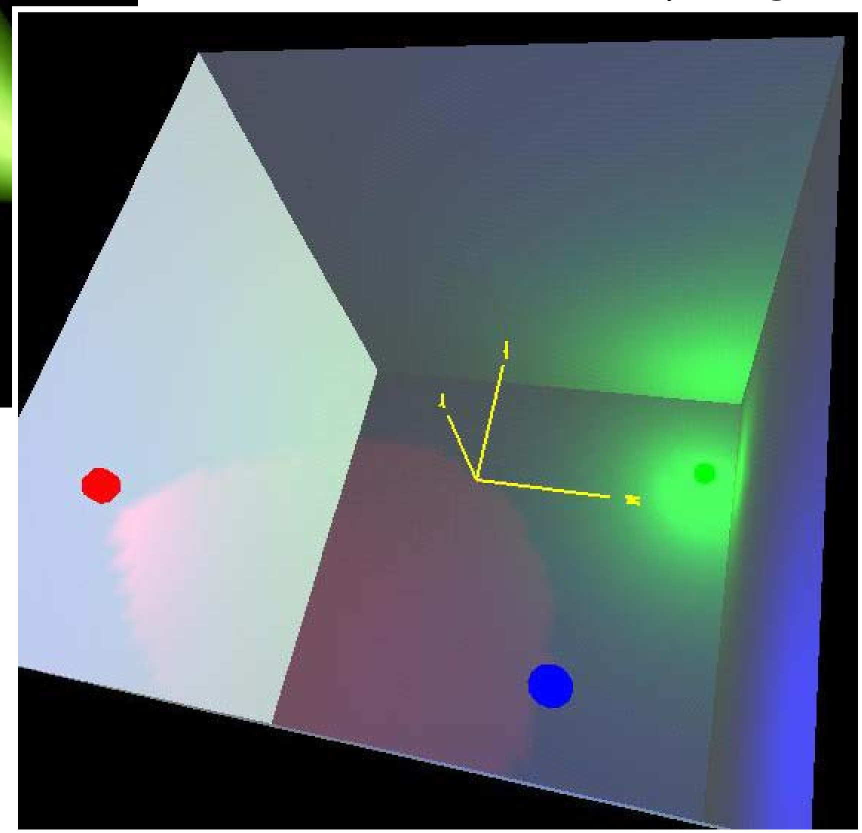
=

Lighting Examples



Omnidirectional Point Light at the Eye

Spot Lights

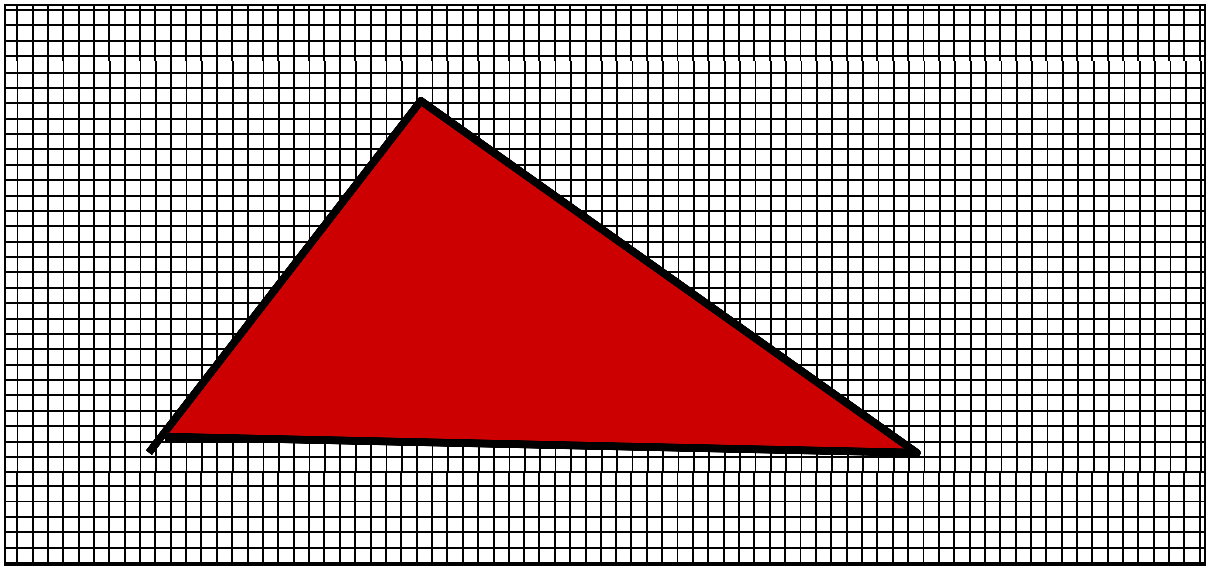


Two Types of Rendering

1. Starts at the object
2. Starts at the eye

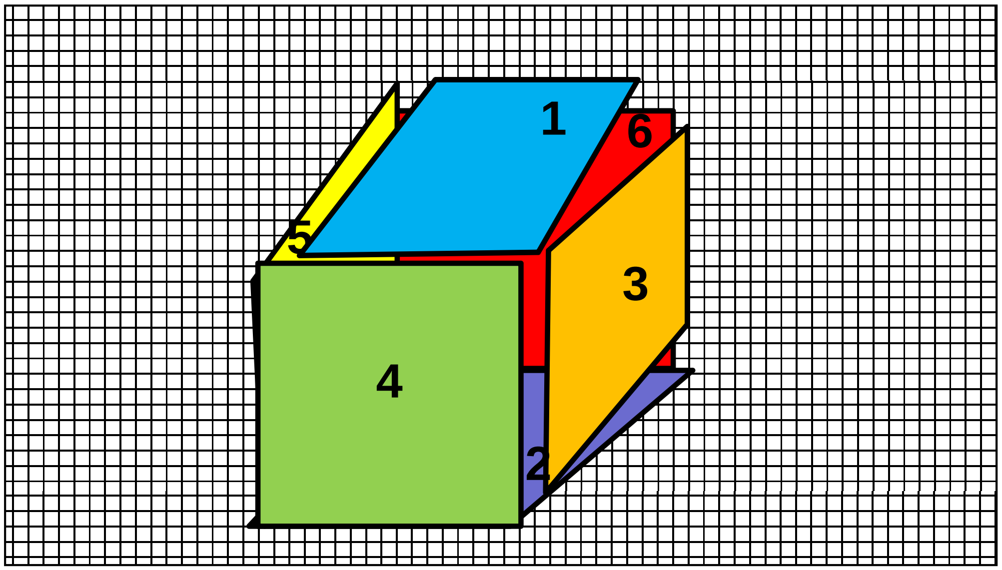
Starts at the Object

This is the typical kind of rendering you get on a graphics card.
Start with the geometry and project it onto the pixels.

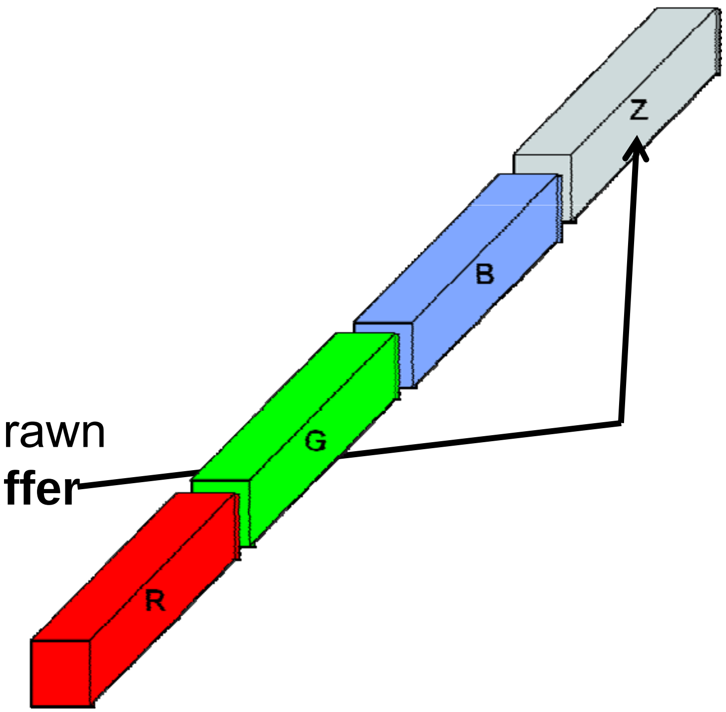


How do things in front look like they are *really* in front?

Your application might draw the polygons in 1-2-3-4-5-6 order, but 1, 3, and 4 still need to look like they were drawn last:

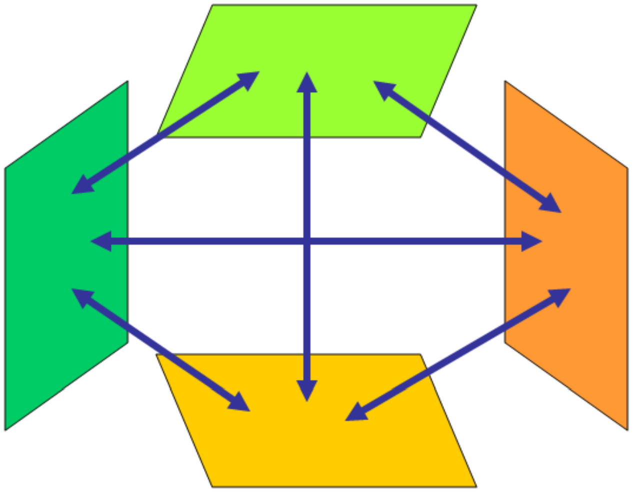


Either the polygons need to be re-arranged to be drawn in a back-to-front order, or we need to have a **Z-buffer**



Another From-the-Object Method -- Radiosity

Based on the idea that all surfaces gather light intensity from all other surfaces



The fundamental radiosity equation is an energy balance that says:

"The light energy leaving surface *i* equals the amount of light energy generated by surface *i* plus surface *i*'s reflectivity times the amount of light energy arriving from all other surfaces"

$$B_i A_i = E_i A_i + \rho_i \sum_j B_j A_j F_{j \to i}$$

The Radiosity Equation

$$B_i A_i = E_i A_i + \rho_i \sum_j B_j A_j F_{j \to i}$$

B_i is the light energy intensity shining from surface element *i*

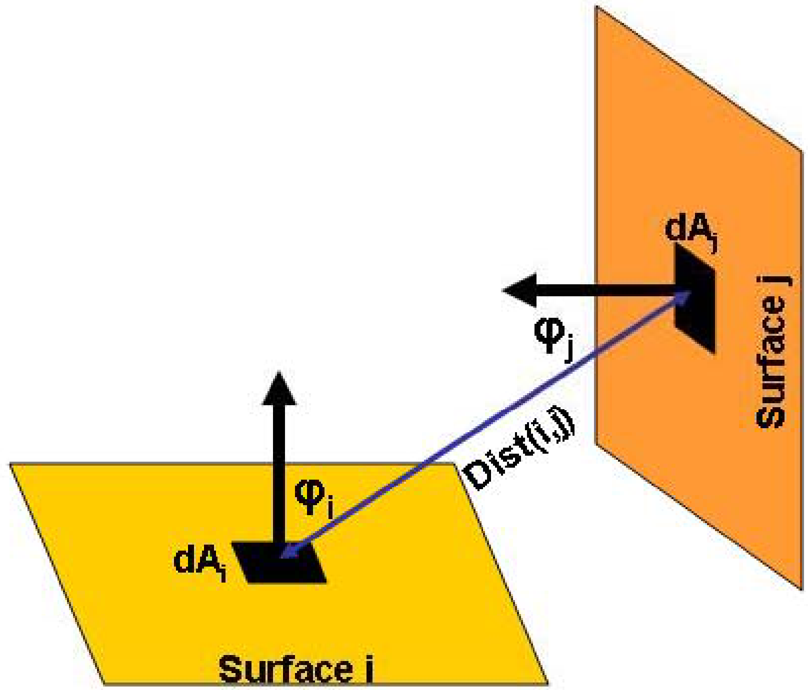
A_i is the area of surface element *i*

E_i is the internally-generated light energy intensity for surface element *i*

ρ_i is surface element *i*'s reflectivity

$F_{j \to i}$ is referred to as the Form Factor, or Shape Factor, and describes what percent of the energy leaving surface element *j* that arrives at surface element *i*

The Radiosity Shape Factor



$$F_{j \to i} = \int_{Ai} \int_{A_j} visibility(di, dj) \frac{\cos\Theta_i \cos\Theta_j}{\pi Dist(di, dj)^2} dA_j dA_i$$

The Radiosity Matrix Equation

Expand $B_i A_i = E_i A_i + \rho_i \sum_j B_j A_j F_{j \to i}$

For each surface element, and re-arrange to solve for the surface intensities, the *B*'s:

$$\begin{bmatrix} 1-\rho_1 F_{1\to 1} & -\rho_1 F_{1\to 2} & \bullet\bullet\bullet & -\rho_1 F_{1\to N} \\ -\rho_2 F_{2\to 1} & 1-\rho_2 F_{2\to 2} & \bullet\bullet\bullet & -\rho_2 F_{2\to N} \\ \bullet\bullet\bullet & \bullet\bullet\bullet & \bullet\bullet\bullet & \bullet\bullet\bullet \\ -\rho_N F_{N\to 1} & -\rho_N F_{N\to 2} & \bullet\bullet\bullet & 1-\rho_N F_{N\to N} \end{bmatrix} \begin{Bmatrix} B_1 \\ B_2 \\ \bullet\bullet\bullet \\ B_N \end{Bmatrix} = \begin{Bmatrix} E_1 \\ E_2 \\ \bullet\bullet\bullet \\ E_N \end{Bmatrix}$$

This is a lot of equations!

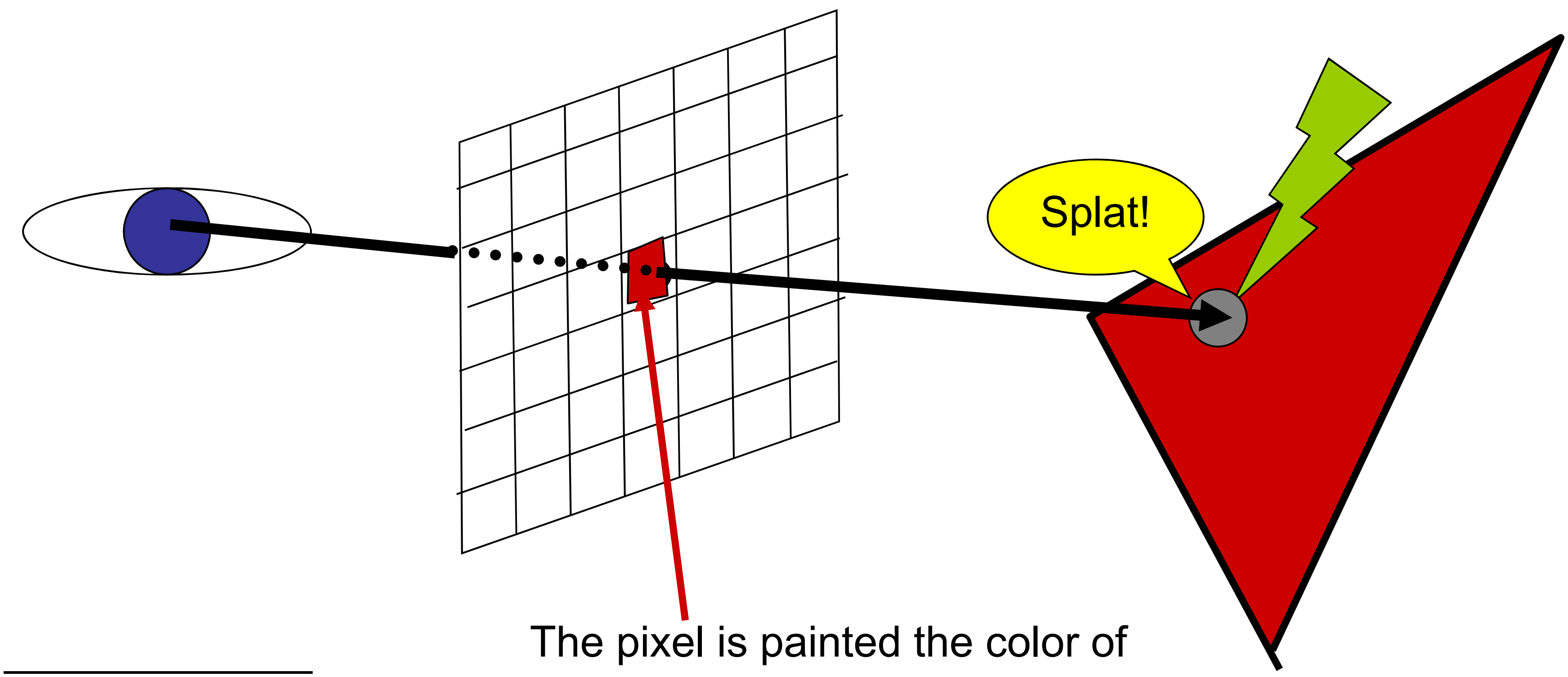
Radiosity Examples

Cornell University

Cornell University

Starts at the Eye

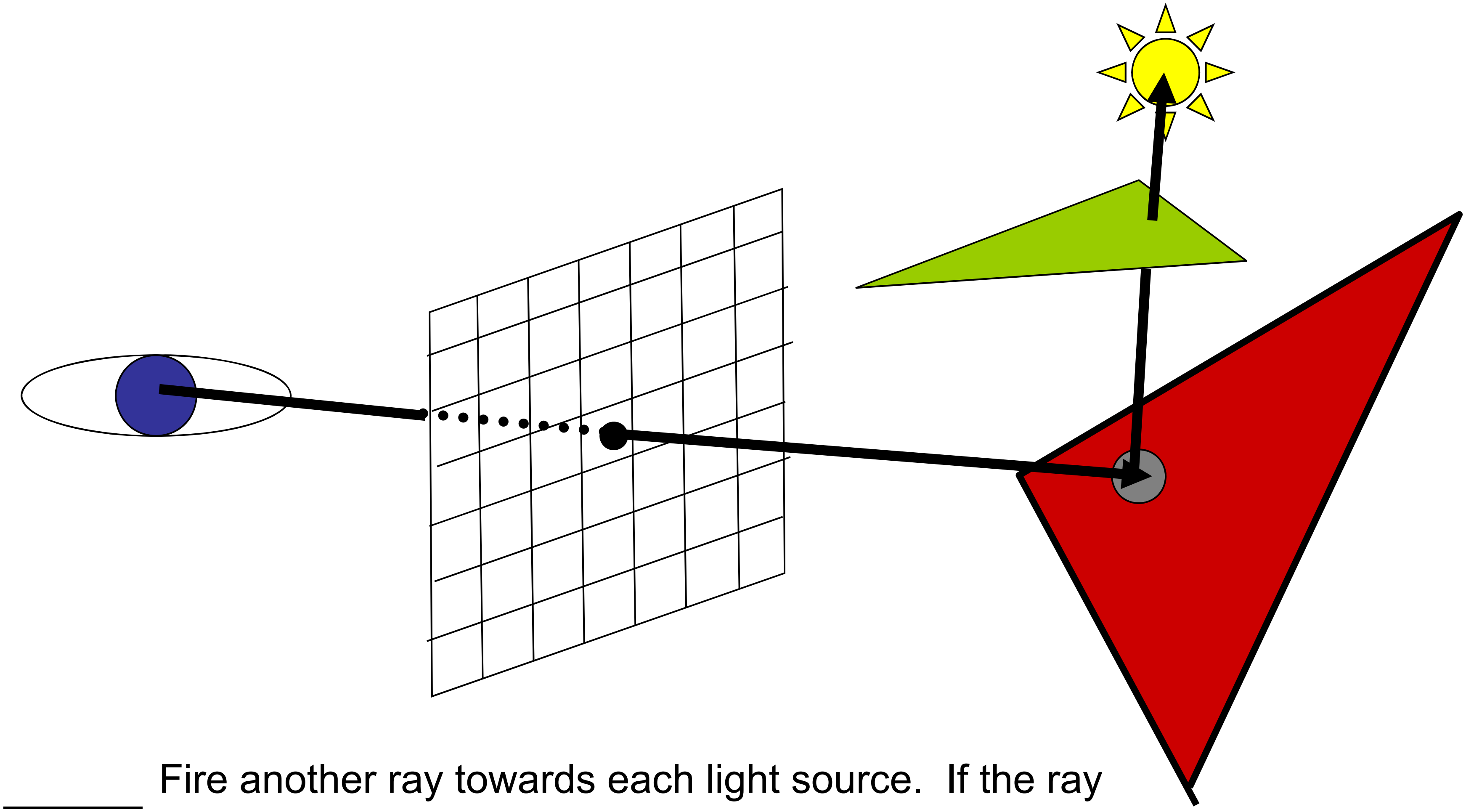
The most common approach in this category is ray-tracing:



The pixel is painted the color of the nearest object that is hit.

Starts at the Eye

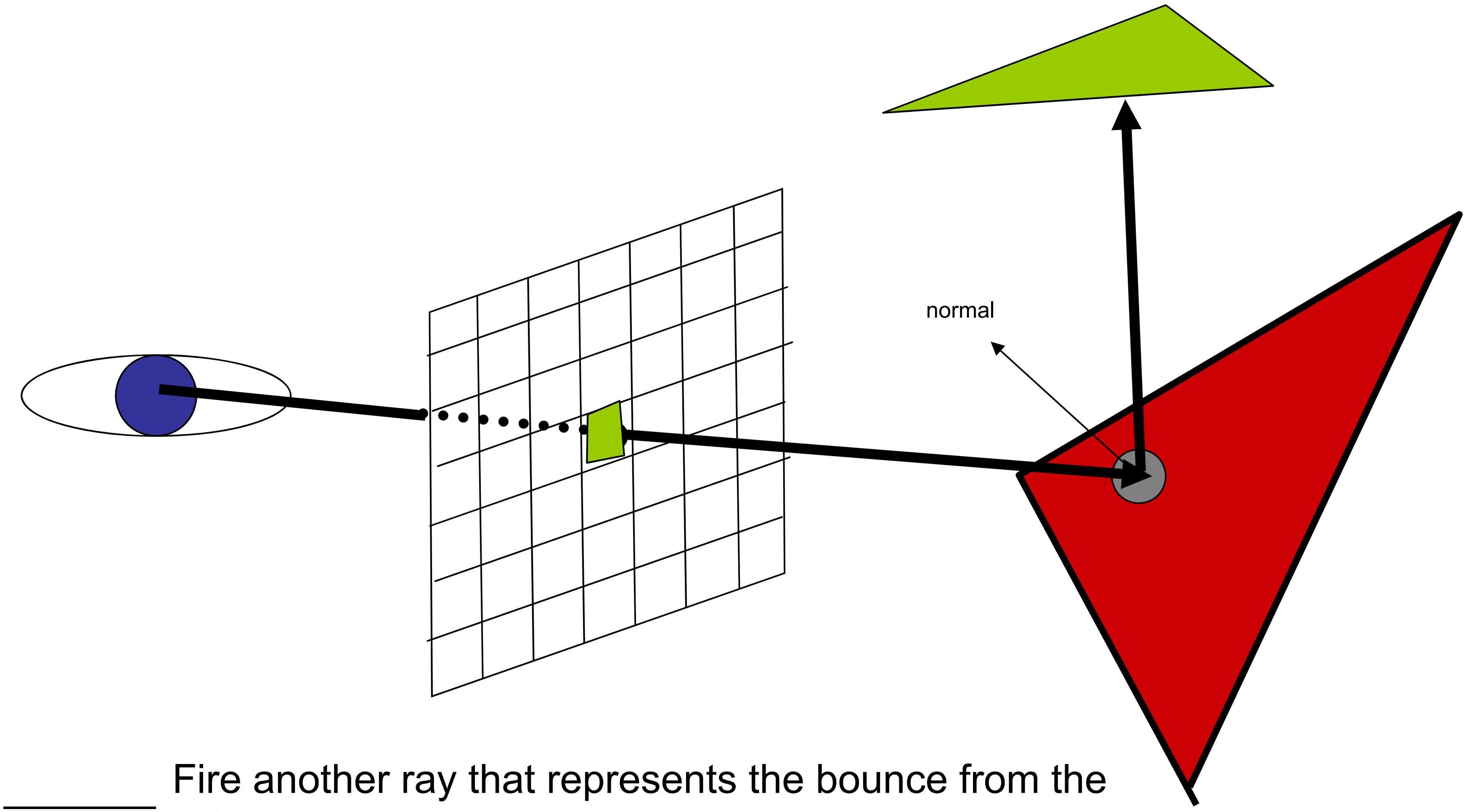
It's also easy to see if this point lies in a shadow:



Fire another ray towards each light source. If the ray hits anything, then the point does not receive that light.

Starts at the Eye

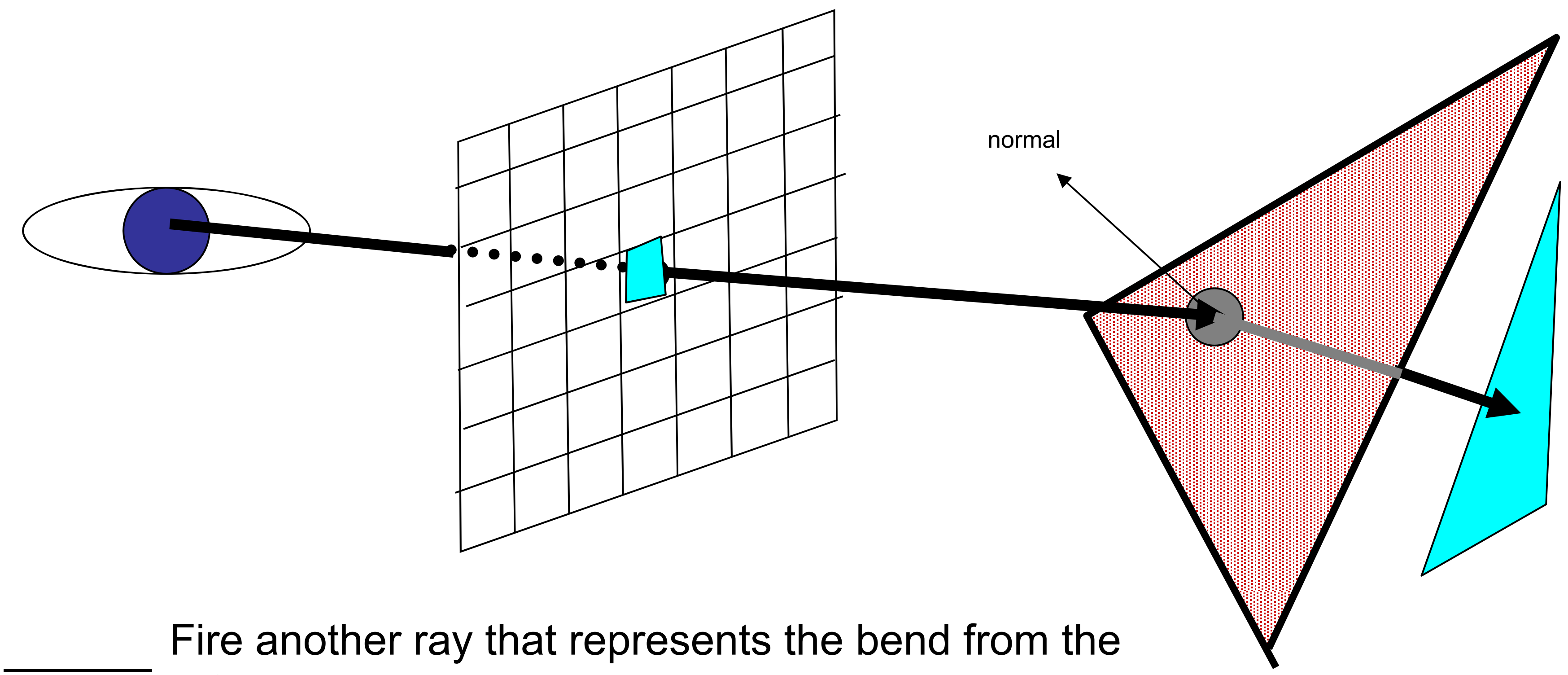
It's also easy to handle reflection



Fire another ray that represents the bounce from the reflection. Paint the pixel the color that this ray sees.

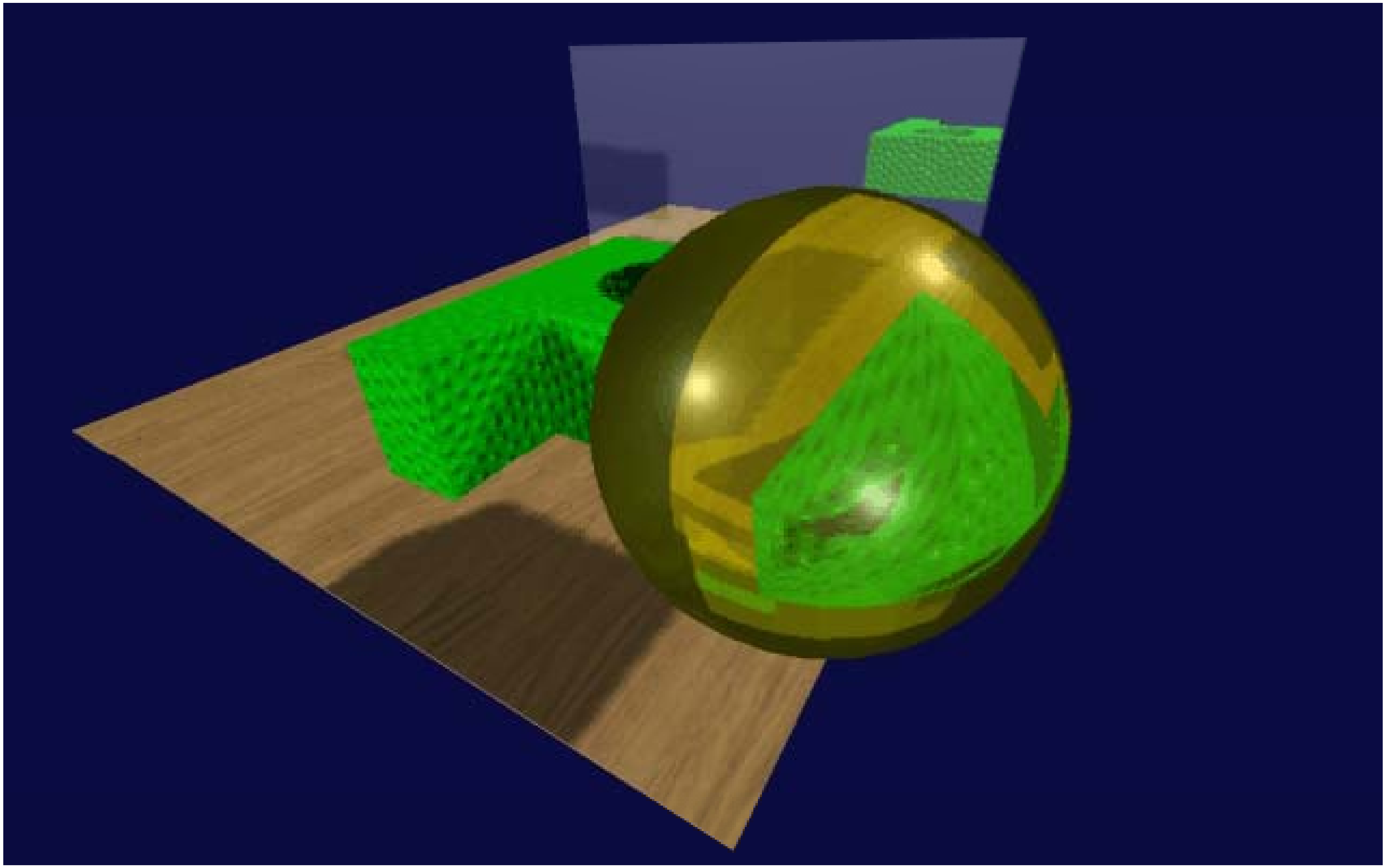
Starts at the Eye

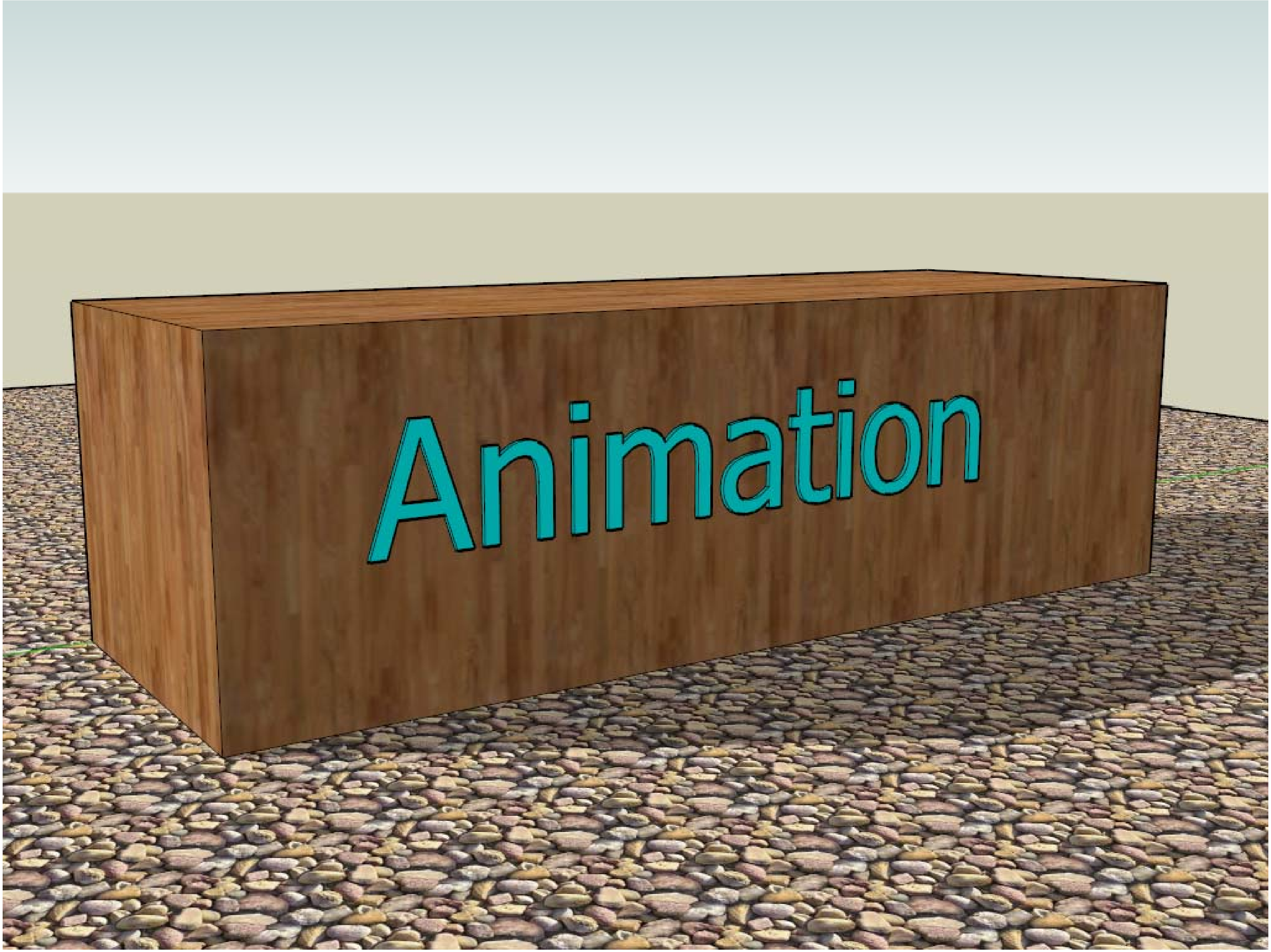
It's also easy to handle refraction



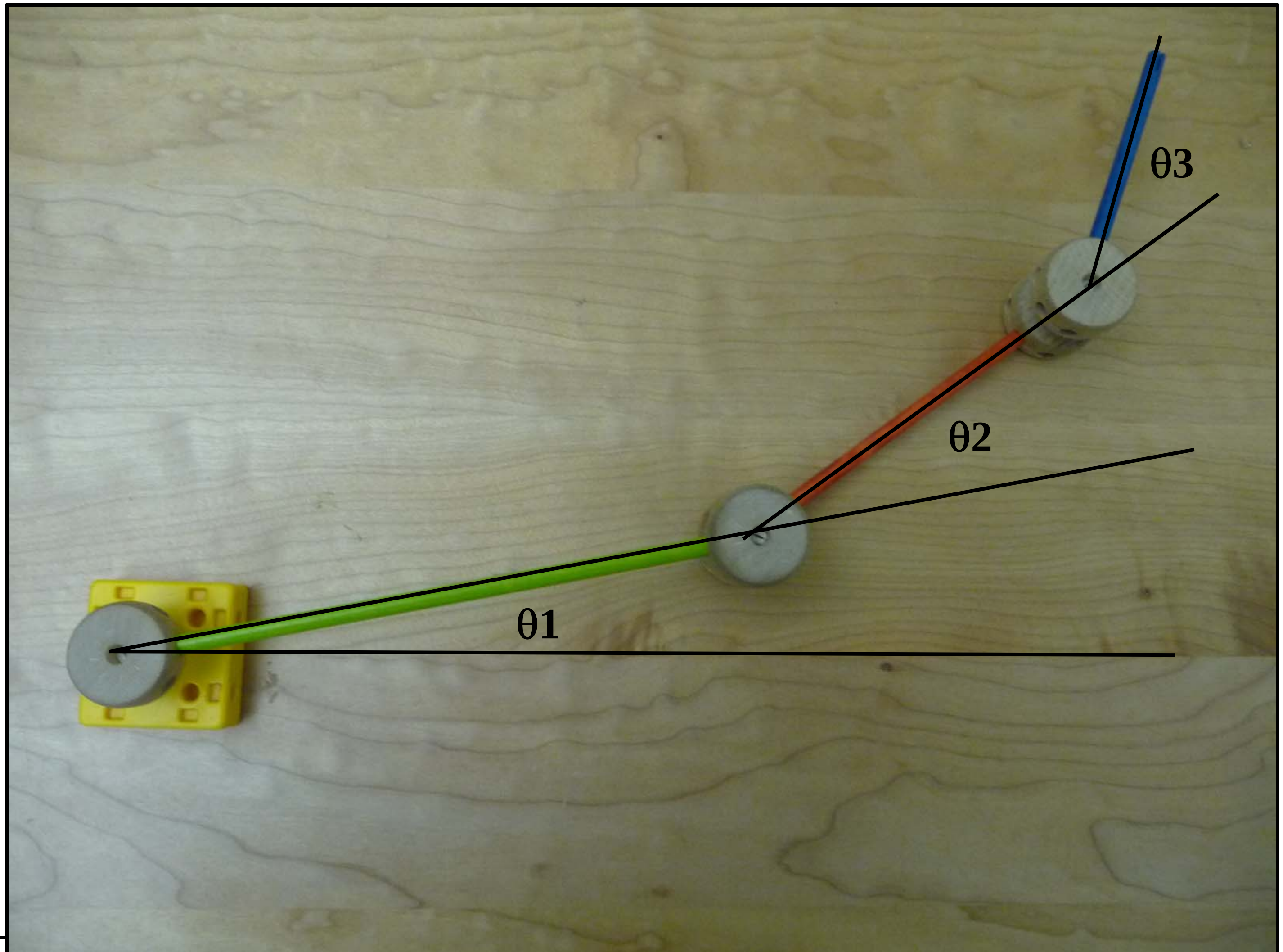
Fire another ray that represents the bend from the refraction. Paint the pixel the color that this ray sees.

Ray Tracing Examples

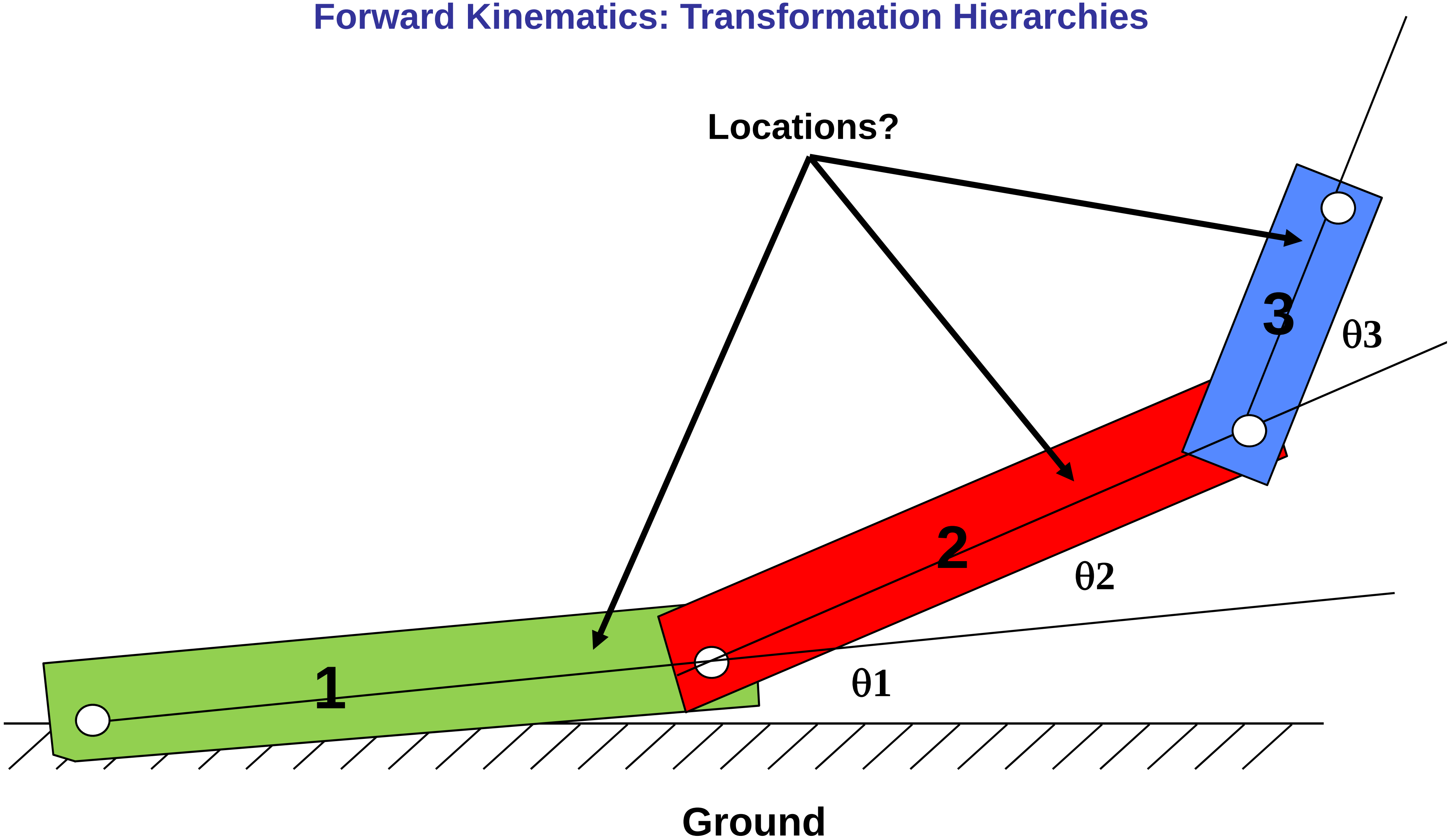



Animation

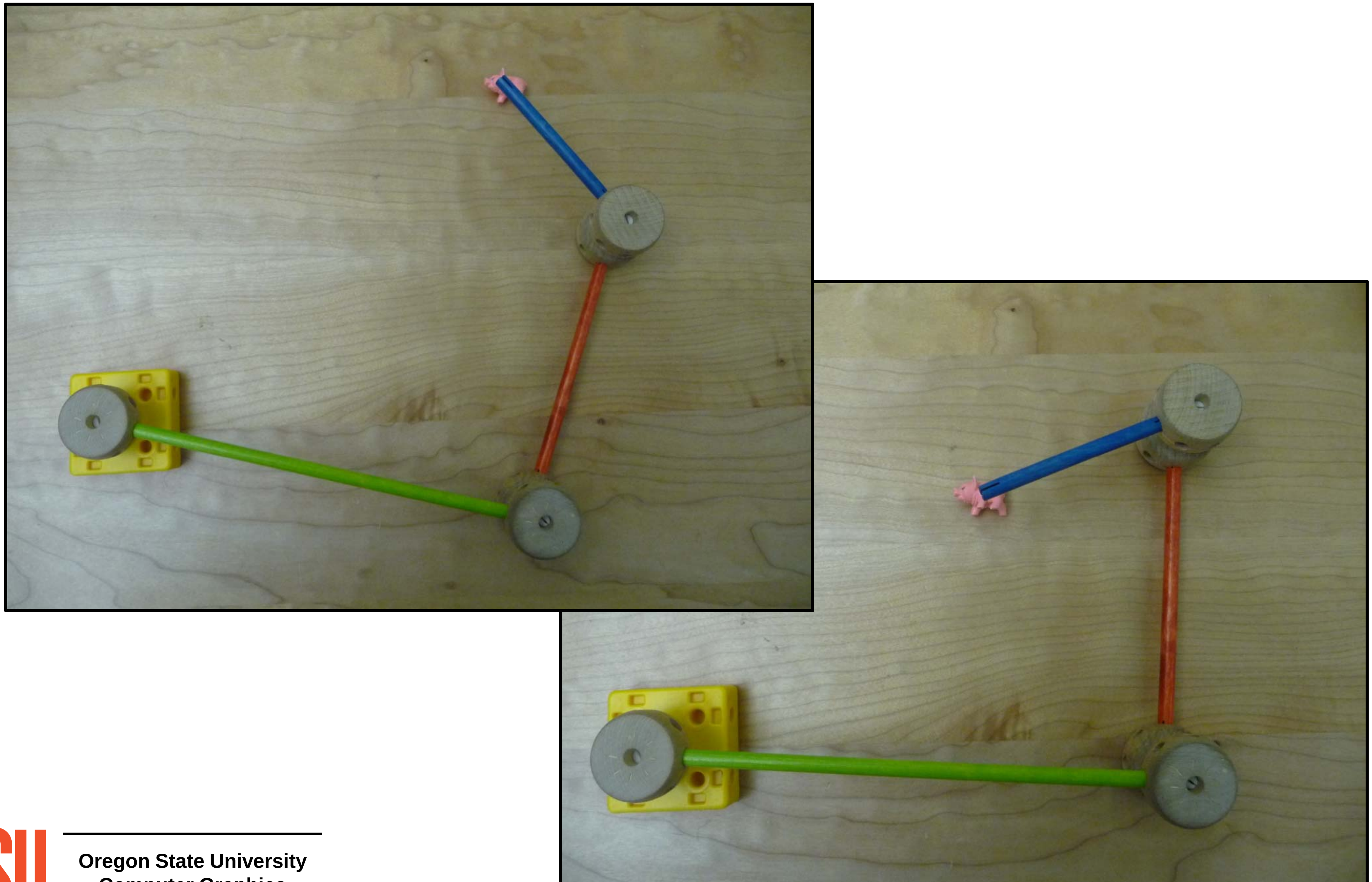
Forward Kinematics: Change Parameters – Things Move (All Children Understand This)



Forward Kinematics: Transformation Hierarchies



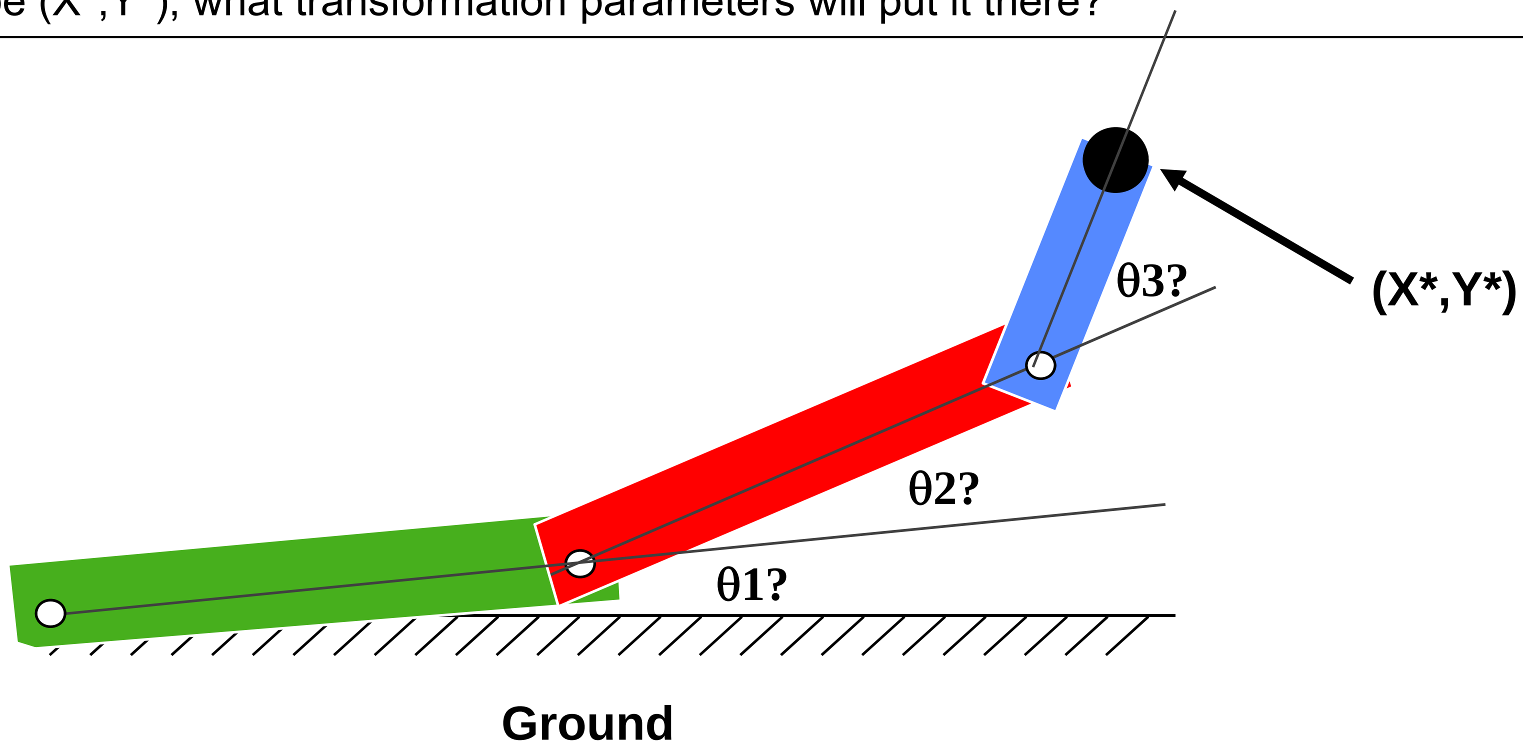
Inverse Kinematics (IK): Things Need to Move – What Parameters Will Make Them Do That?



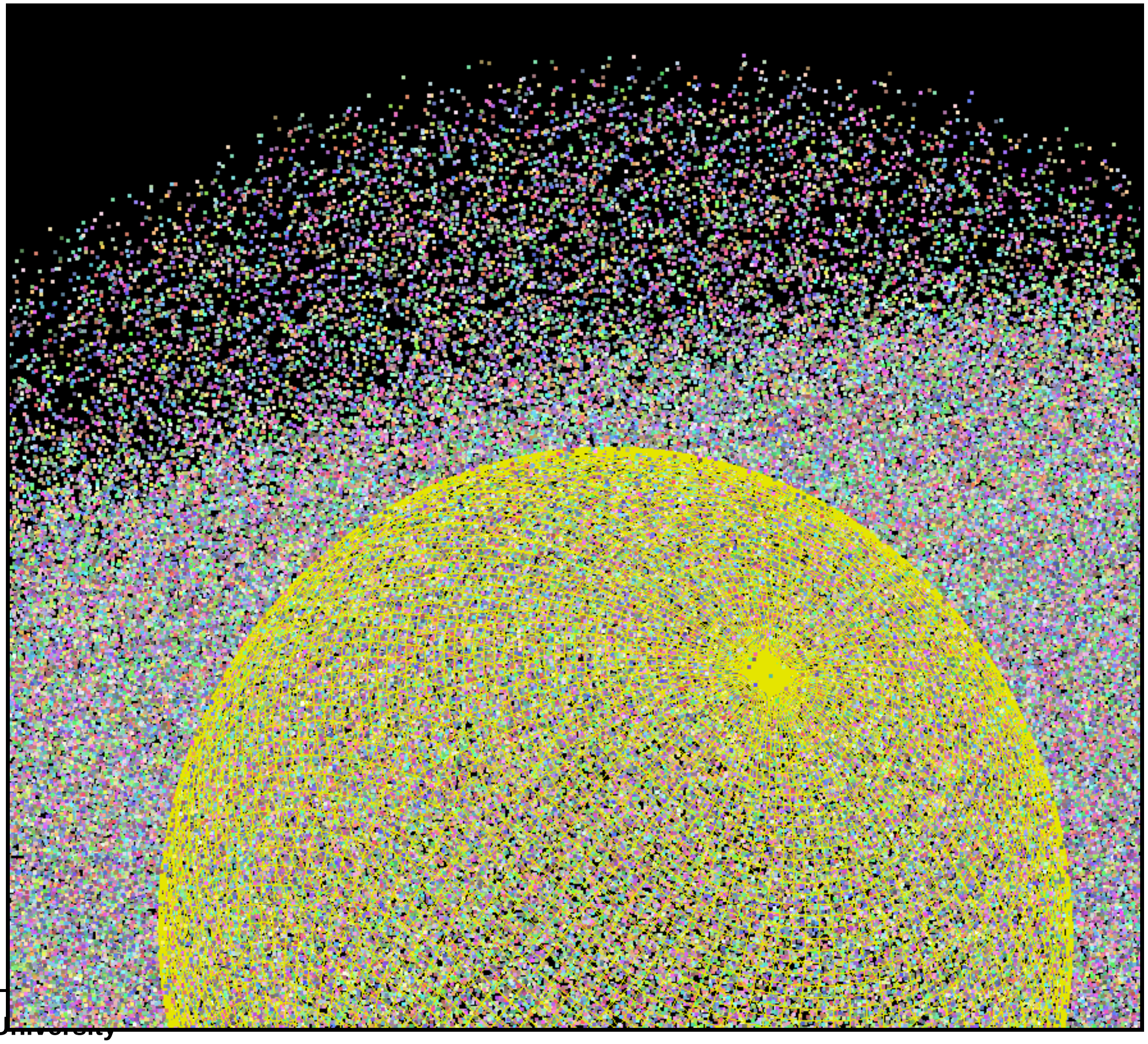
Inverse Kinematics

Forward Kinematics solves the problem “if I know the link transformation parameters, where are the links?”.

Inverse Kinematics (IK) solves the problem “If I know where I want the end of the chain to be (X*,Y*), what transformation parameters will put it there?”

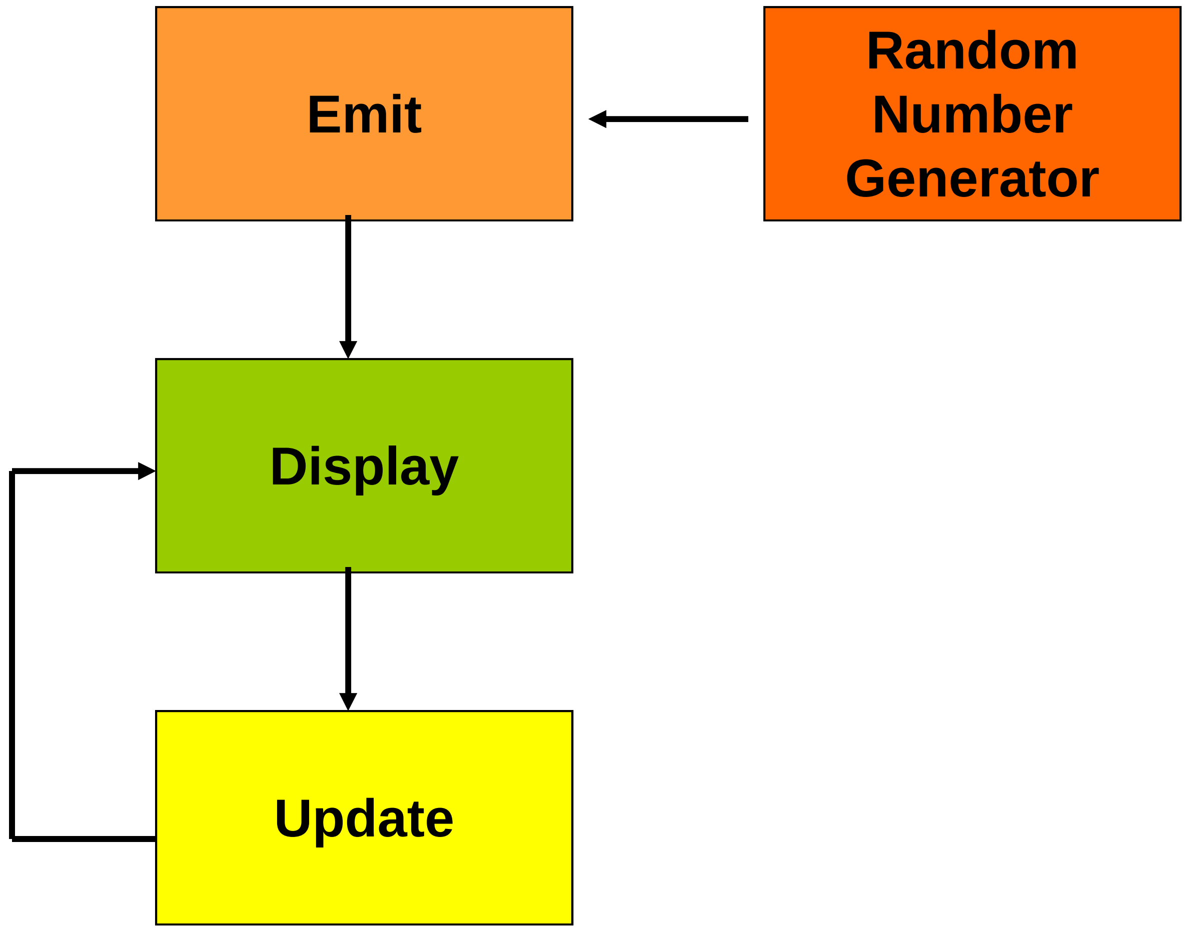


Particle Systems:
A Cross Between Modeling and Animation?

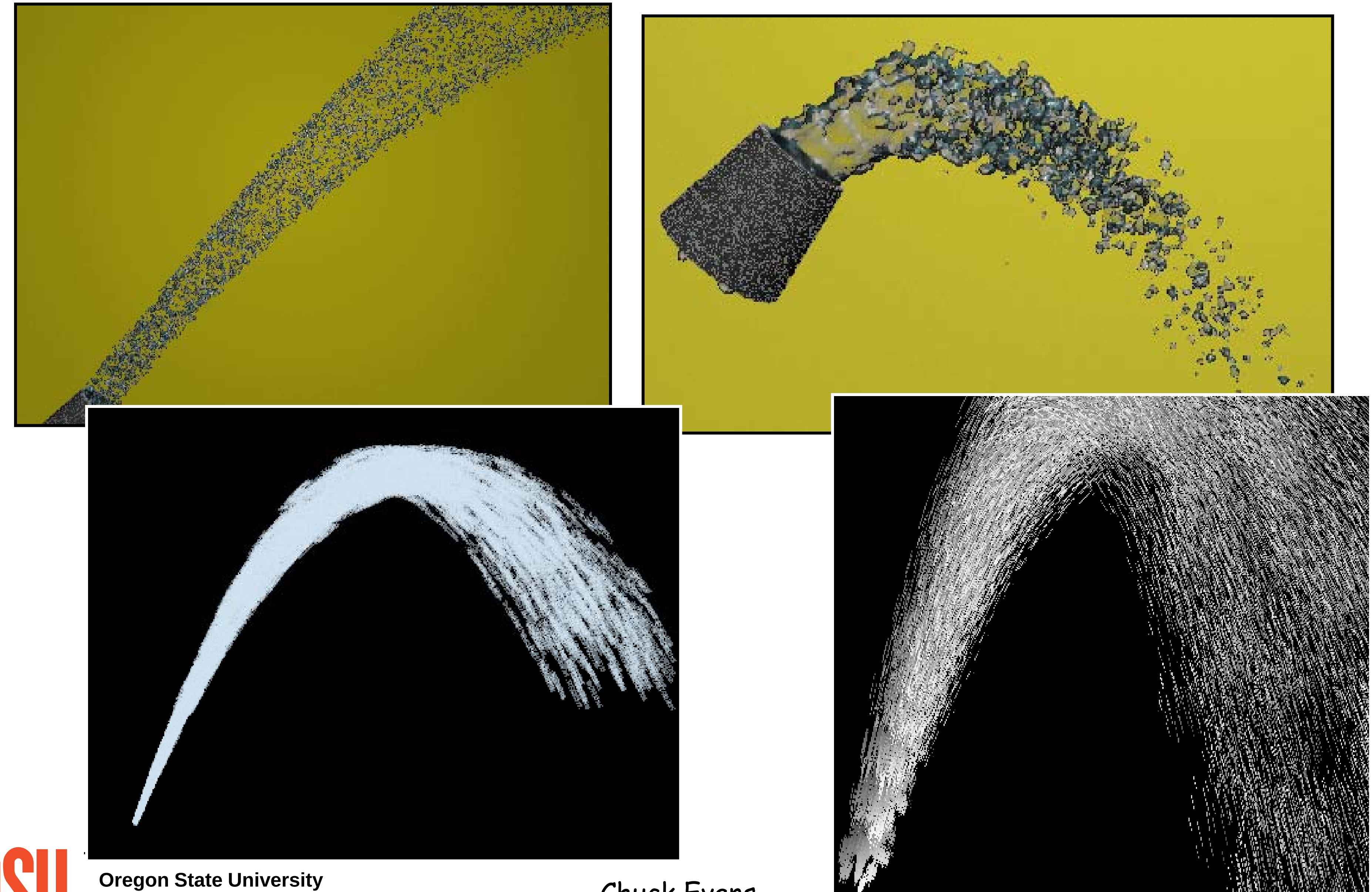


Particle Systems:
A Cross Between Modeling and Animation?

The basic process is:

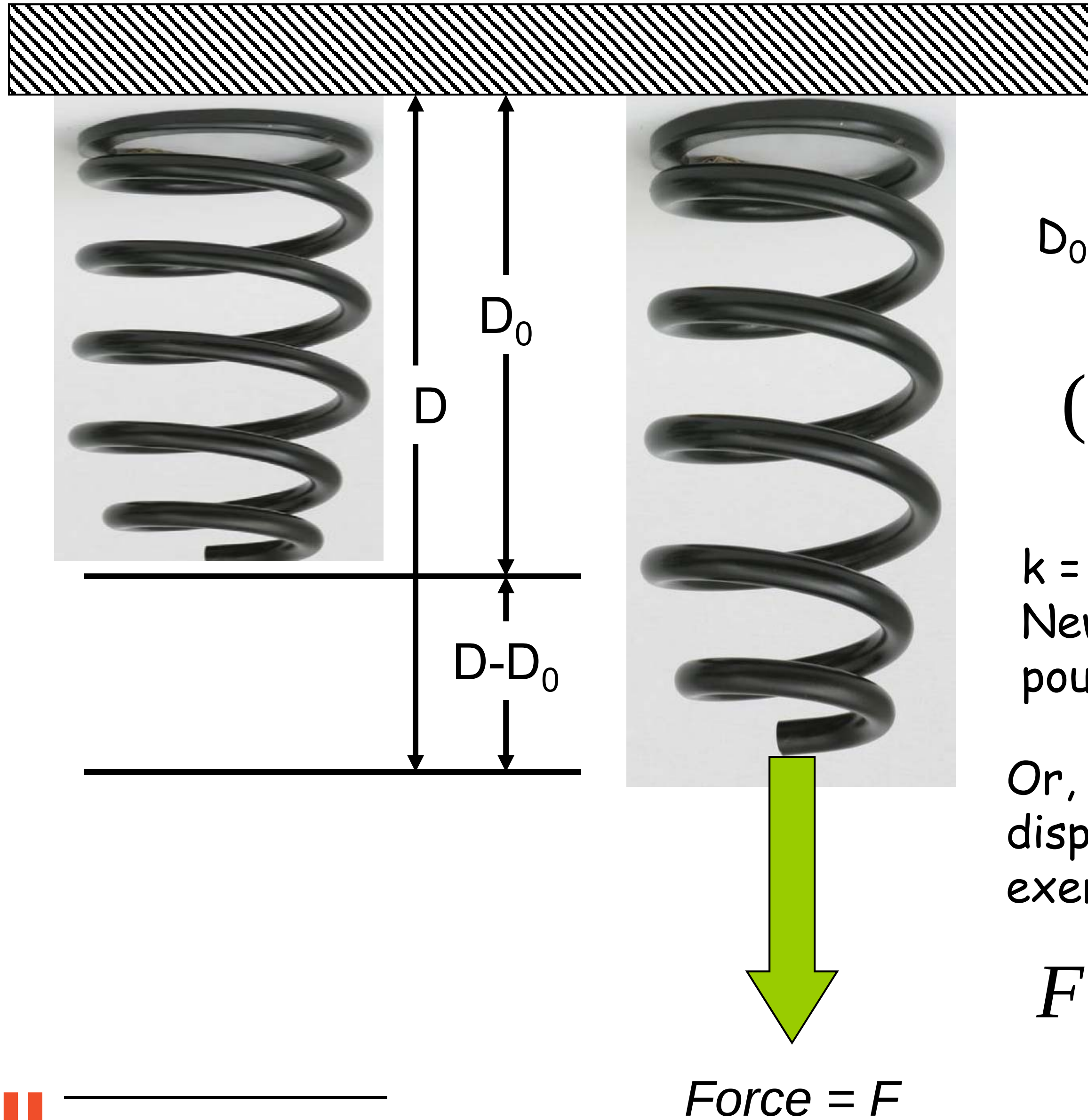


Particle Systems Examples



Chuck Evans

Animating using Physics



D_0 = unloaded spring length

$$(D - D_0) = \frac{F}{k}$$

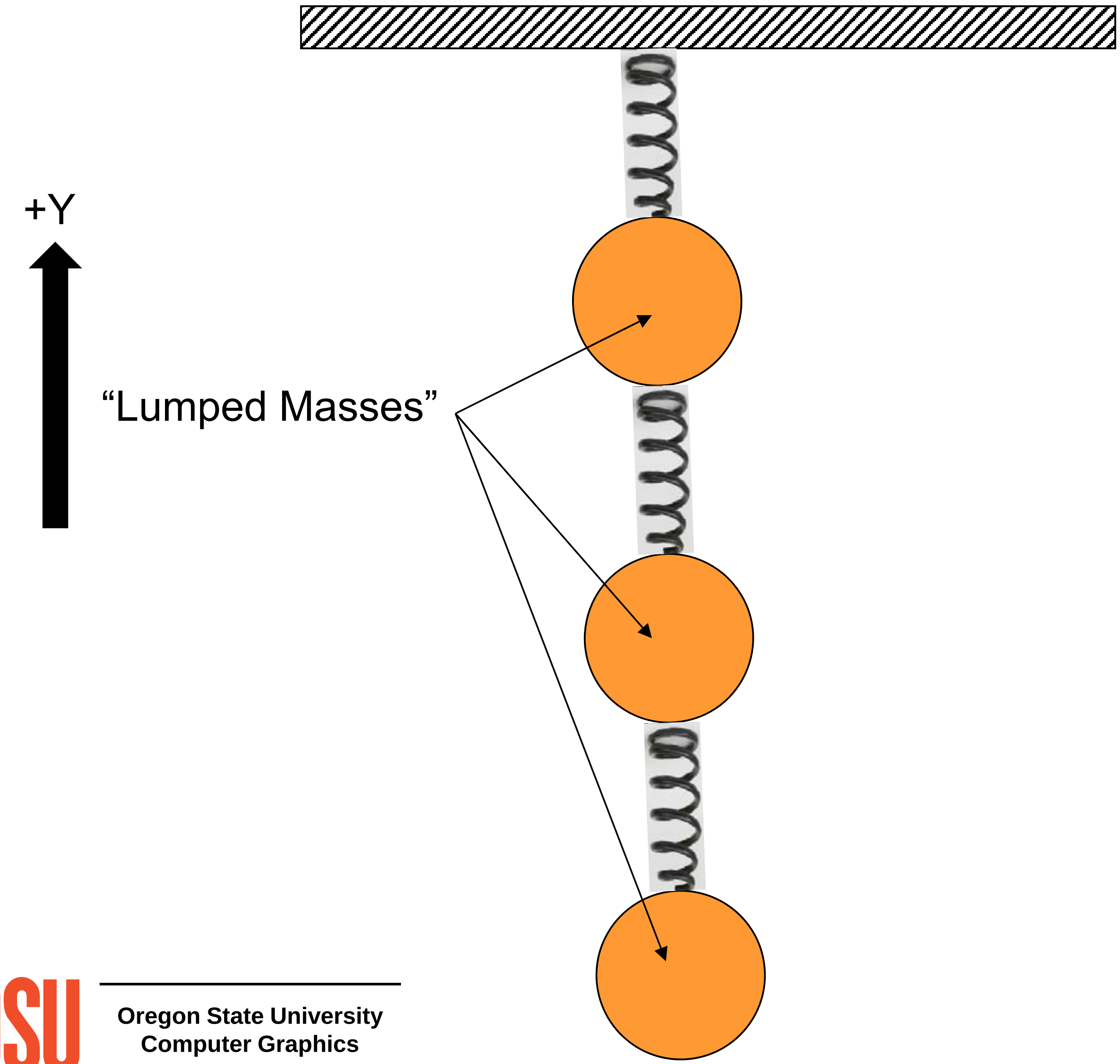
k = ***spring stiffness*** in Newtons/meter or pounds/inch

Or, if you know the displacement, the force exerted by the spring is:

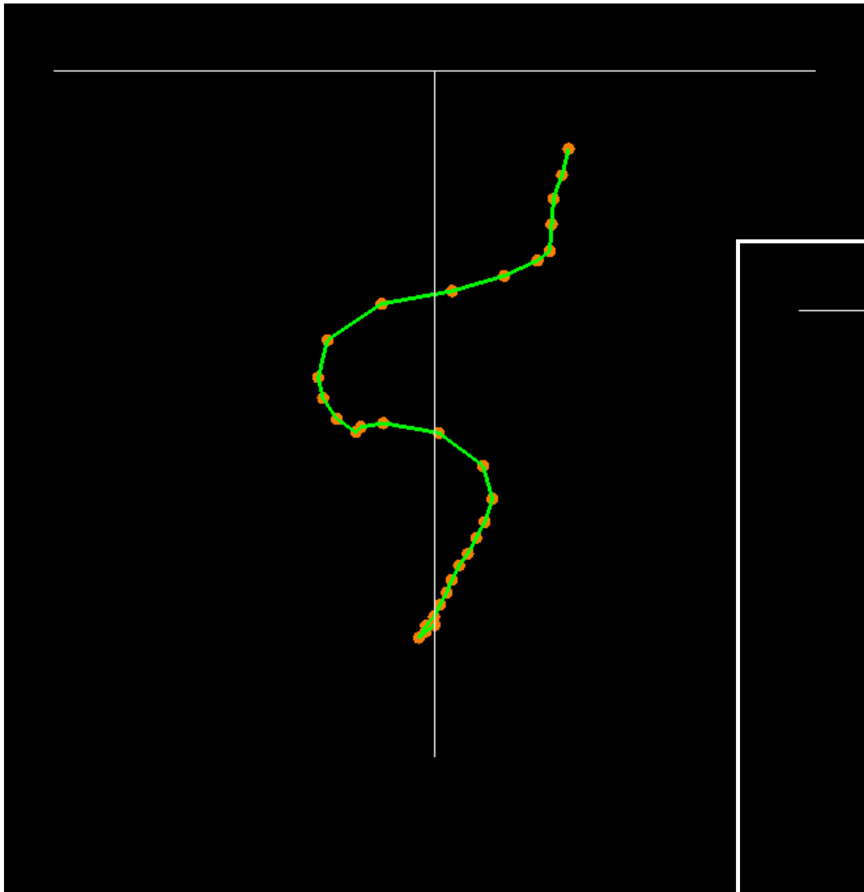
$$F = k\left(D - D_0\right)$$

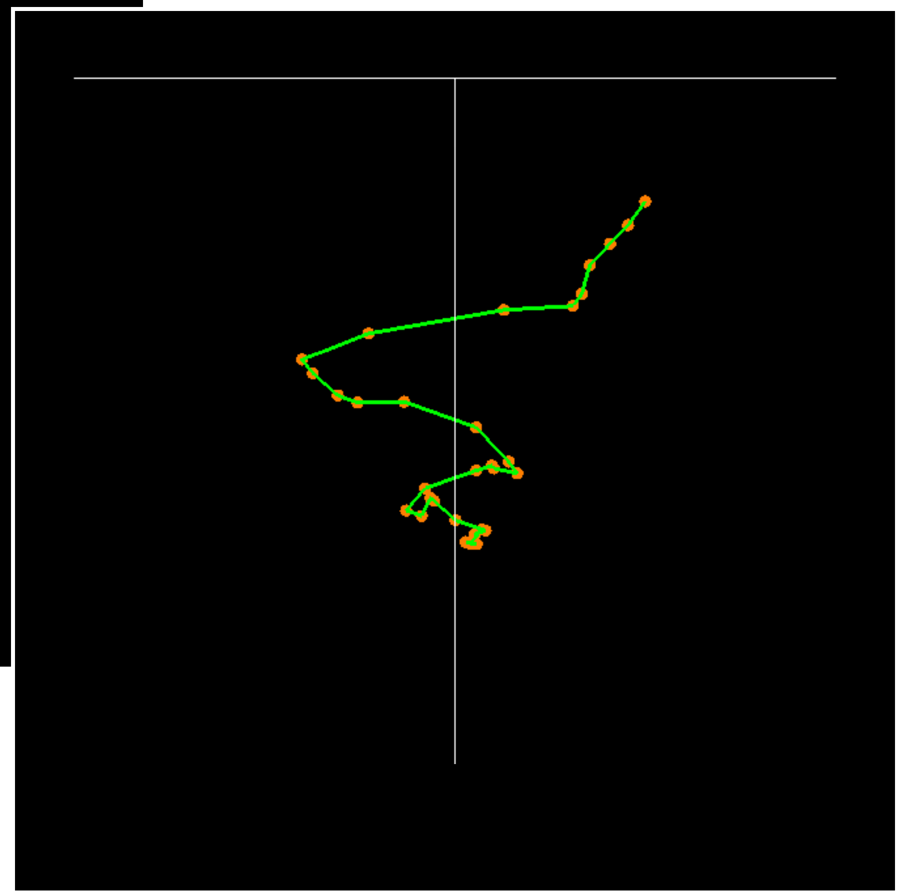
This is known as Hooke's law

Animating using the Physics of a Mesh of Springs

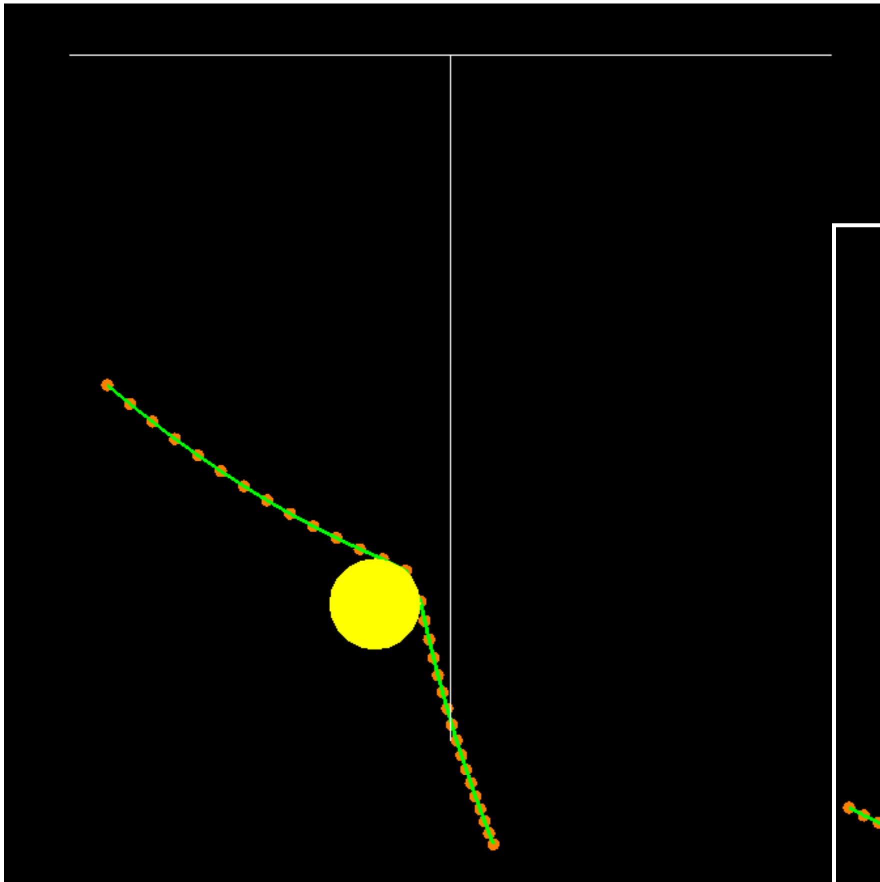


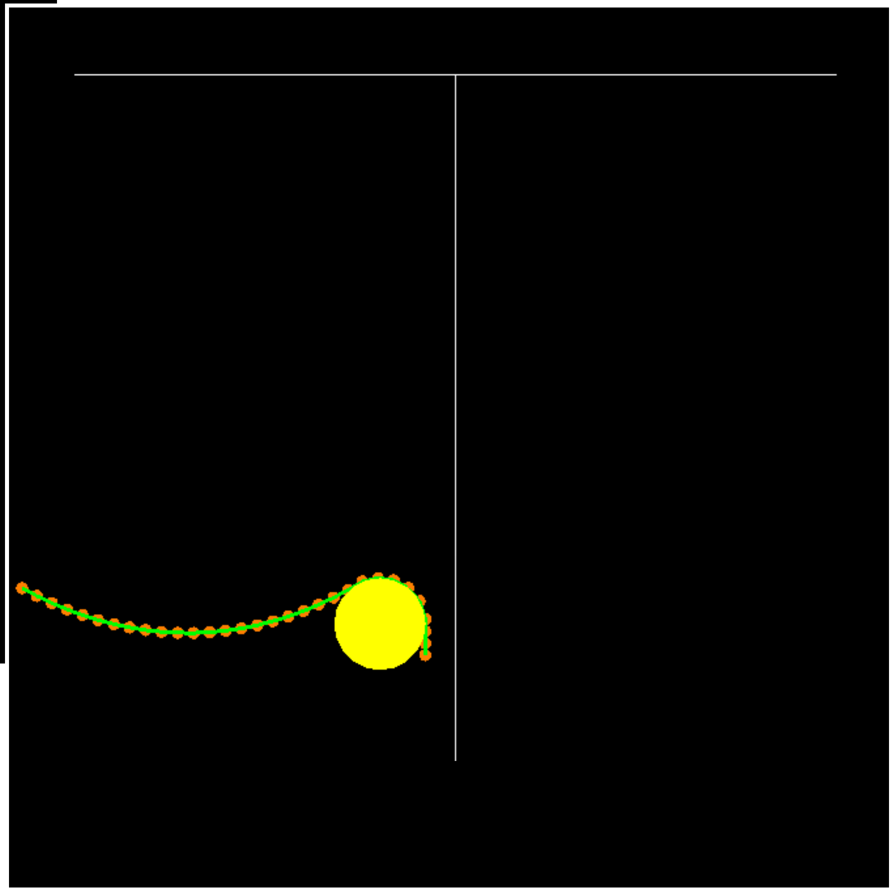
Simulating a Bouncy String



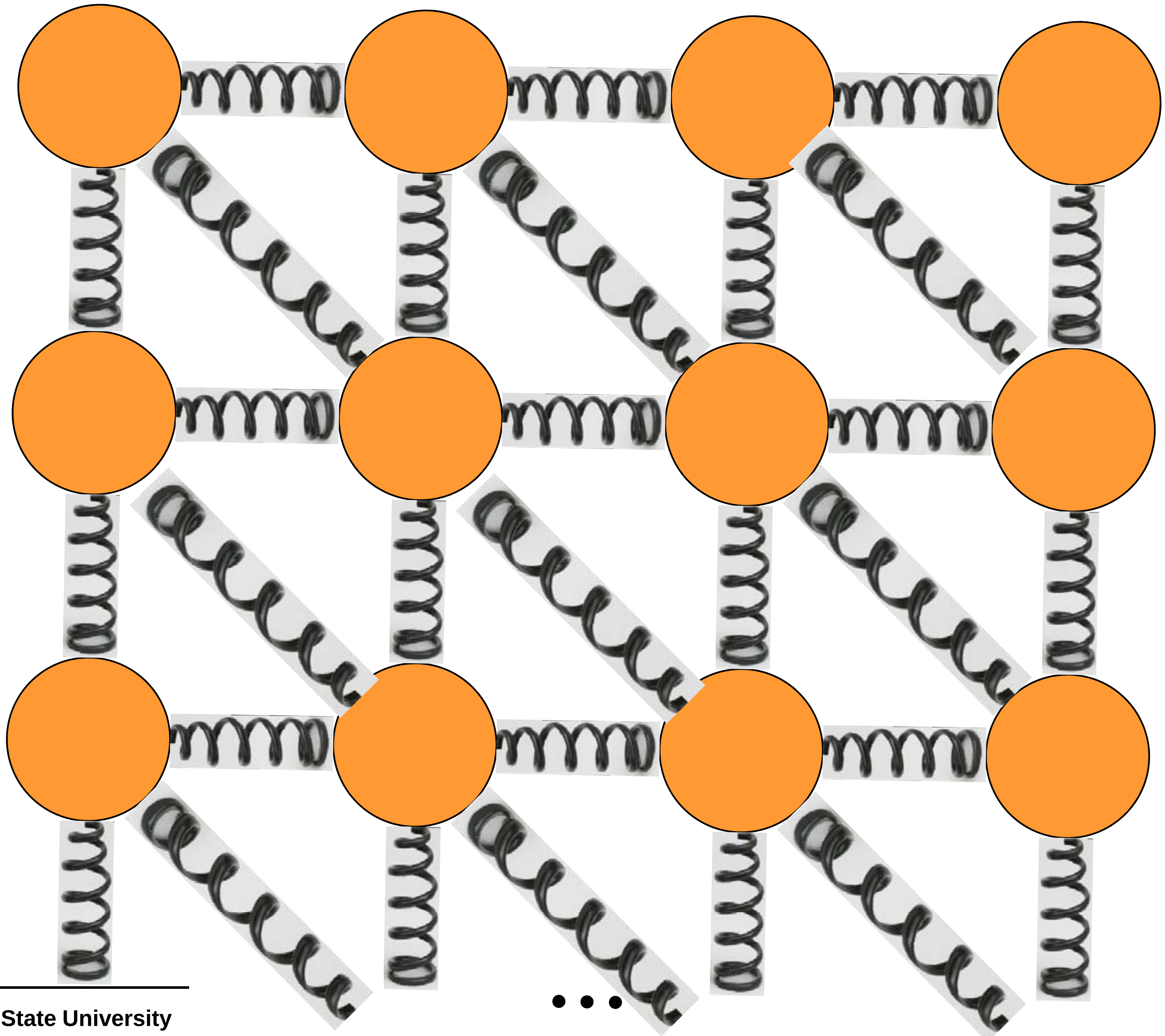


Placing a Physical Barrier in the Scene

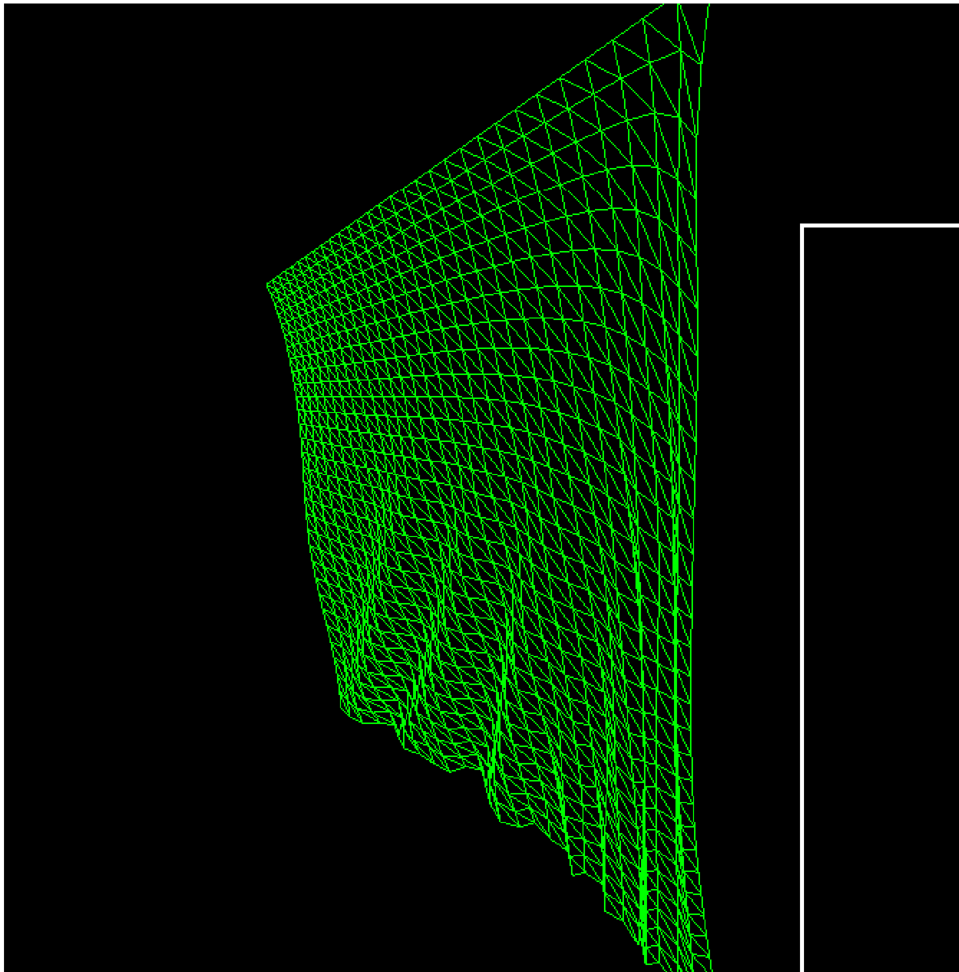


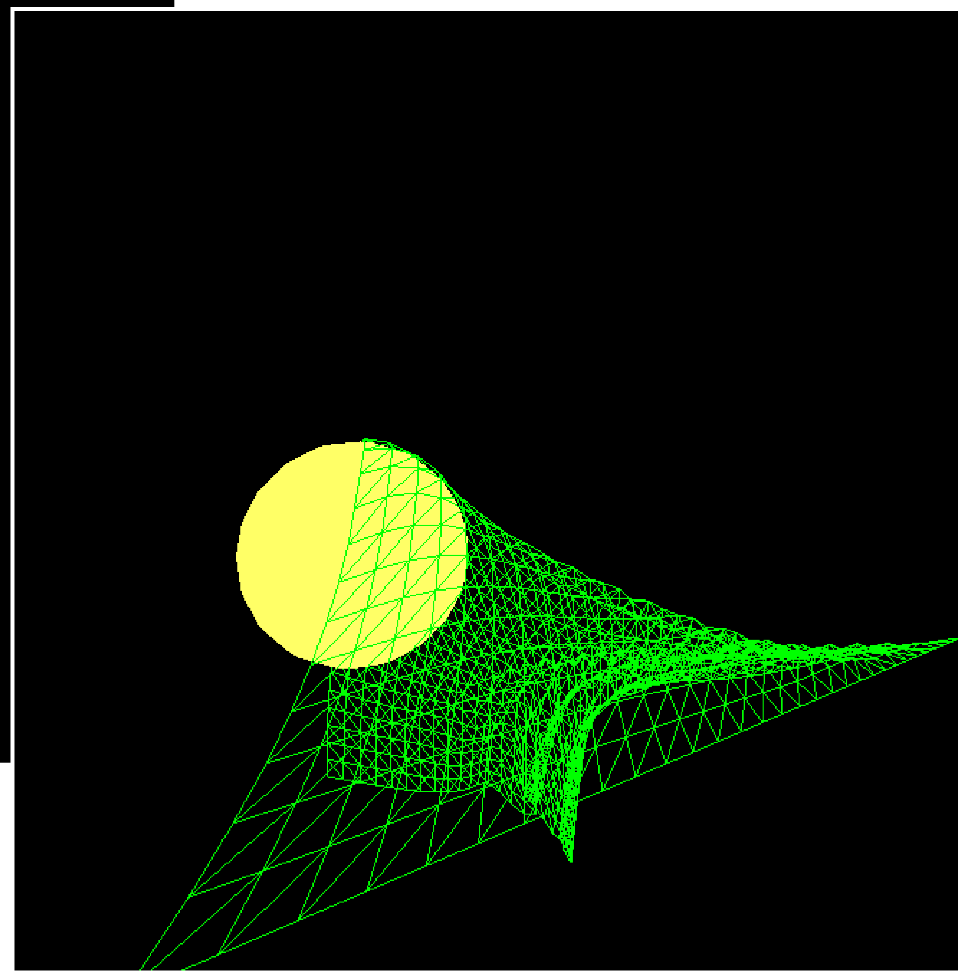


Animating Cloth



Cloth Examples





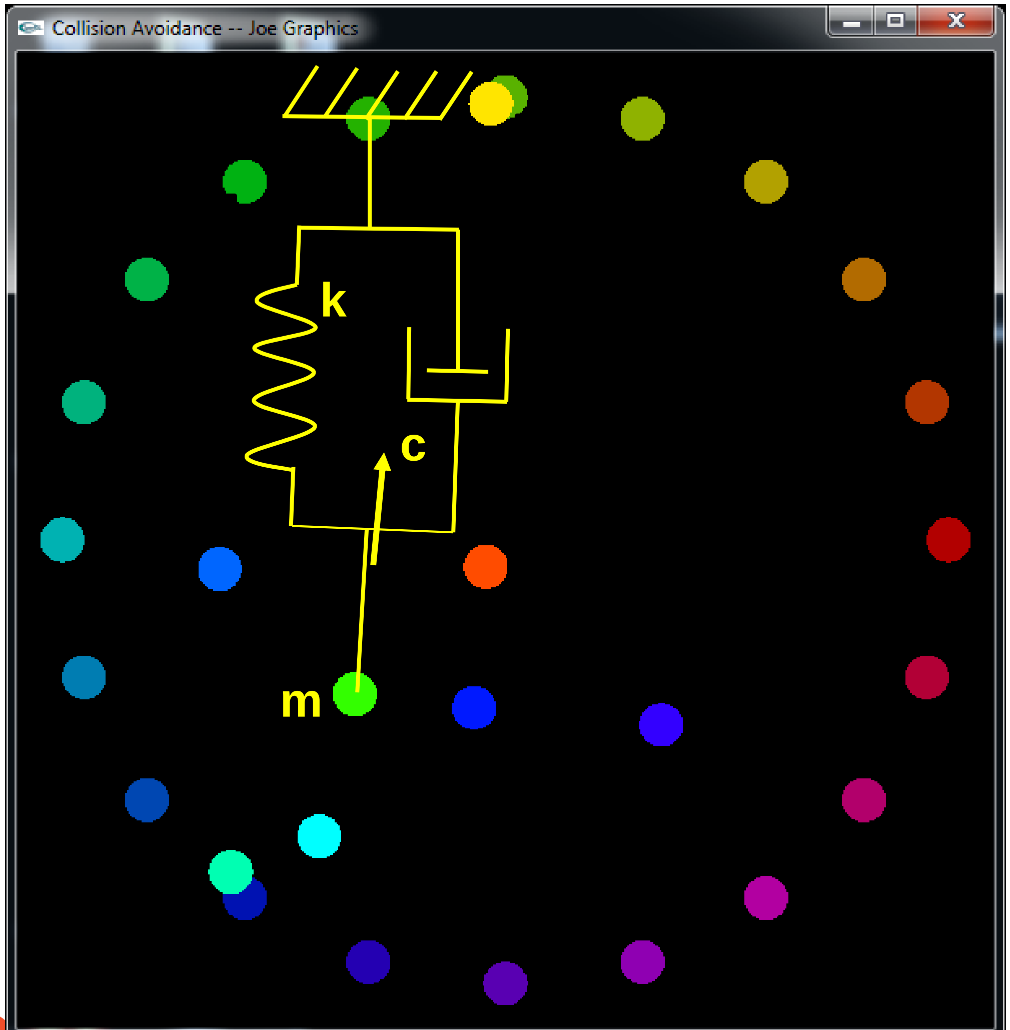
Cloth Examples

David Breen, Donald House, Michael Wozny: *Predicting the Drape of Woven Cloth Using Interacting Particles*

Cloth Examples

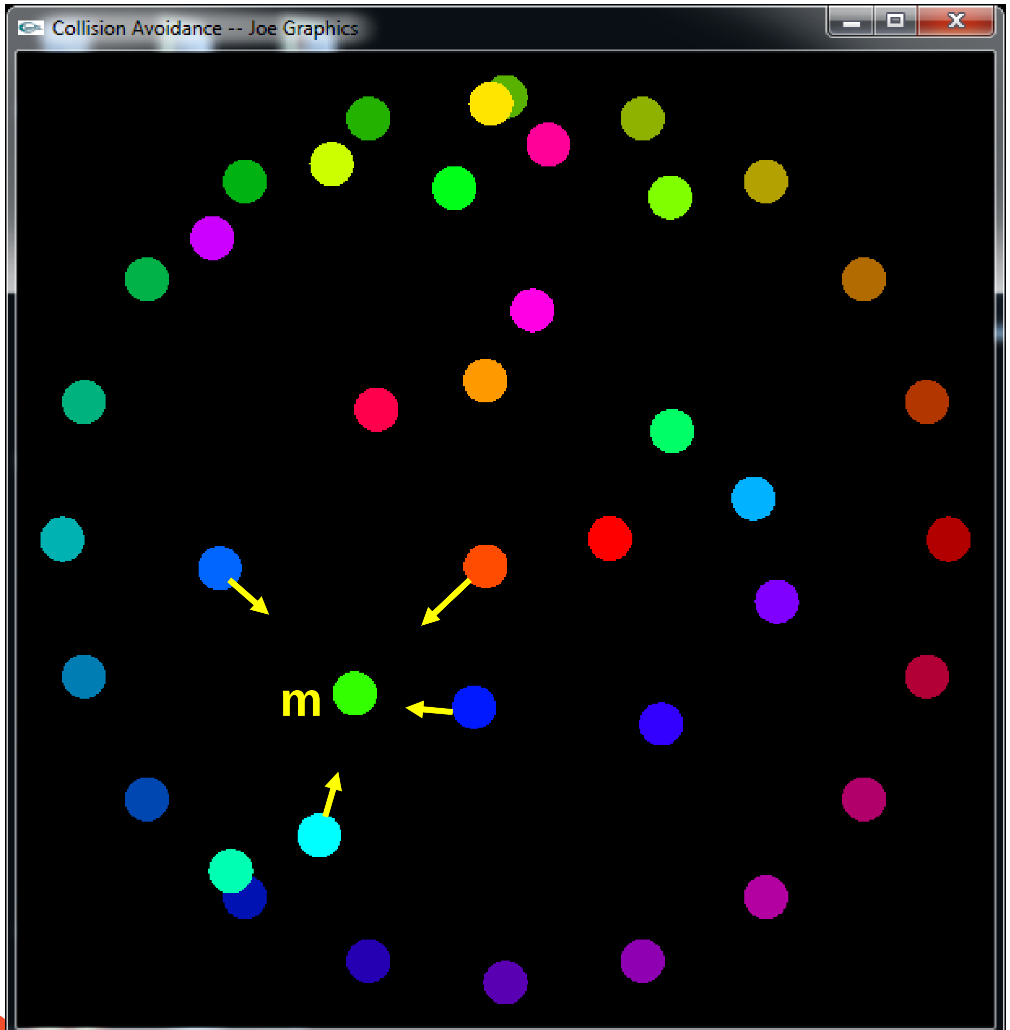
MiraLab, University of Geneva

Functional Animation:
Make the Object Want to Move Towards a Goal Position



$$m\ddot{x} + c\dot{x} + kx = 0$$

Functional Animation:
While Making it Want to Move Away from all other Objects



$$m\ddot{x} = \sum F_{repulsive}$$

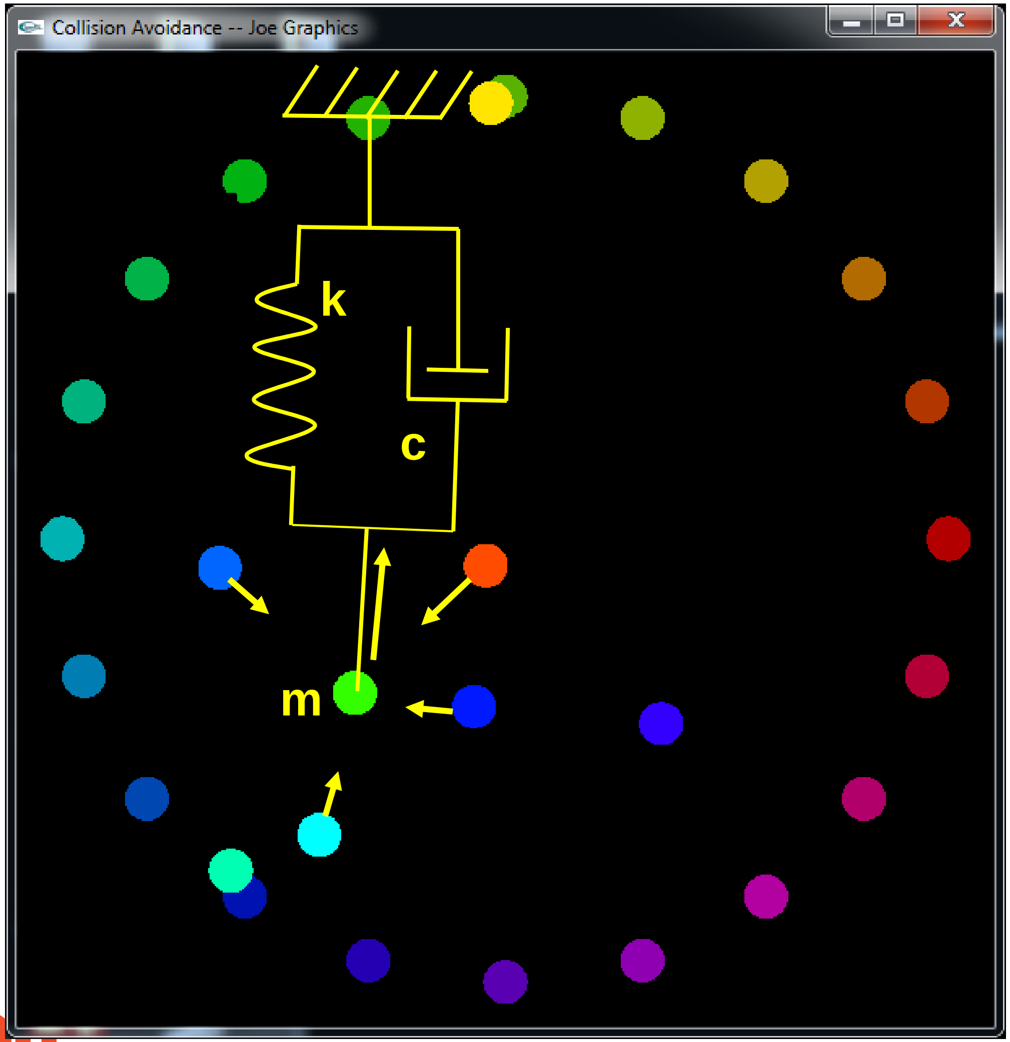
$$F_{repulsive} = \frac{C_{repulse}}{d^{Power}}$$

Repulsion Coefficient: $C_{repulse}$

Distance between the boundaries of the 2 bodies: d

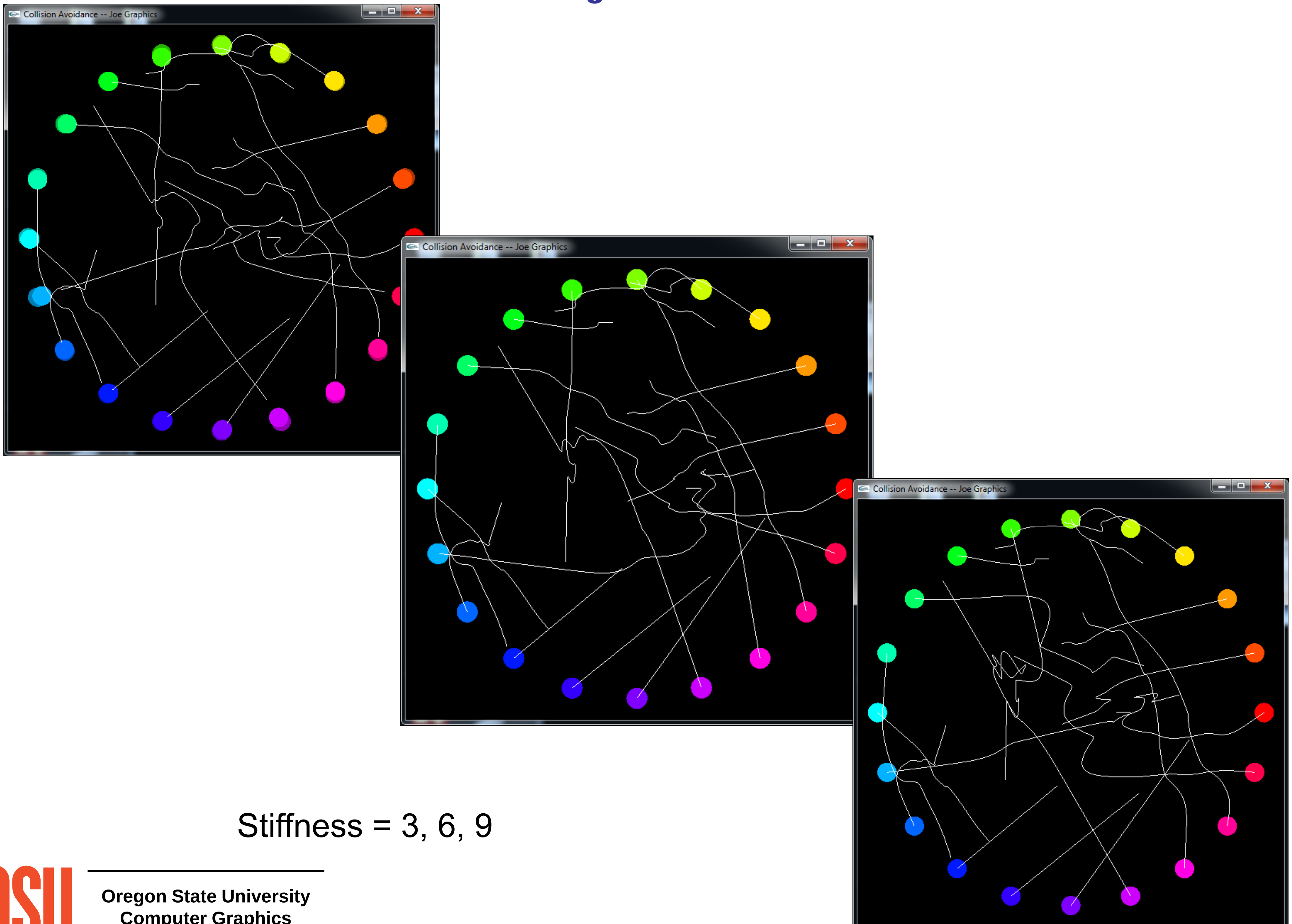
Repulsion Exponent: $Power$

Total Goal – Make the Free Body Move Towards its Final Position While Being Repelled by the Other Bodies



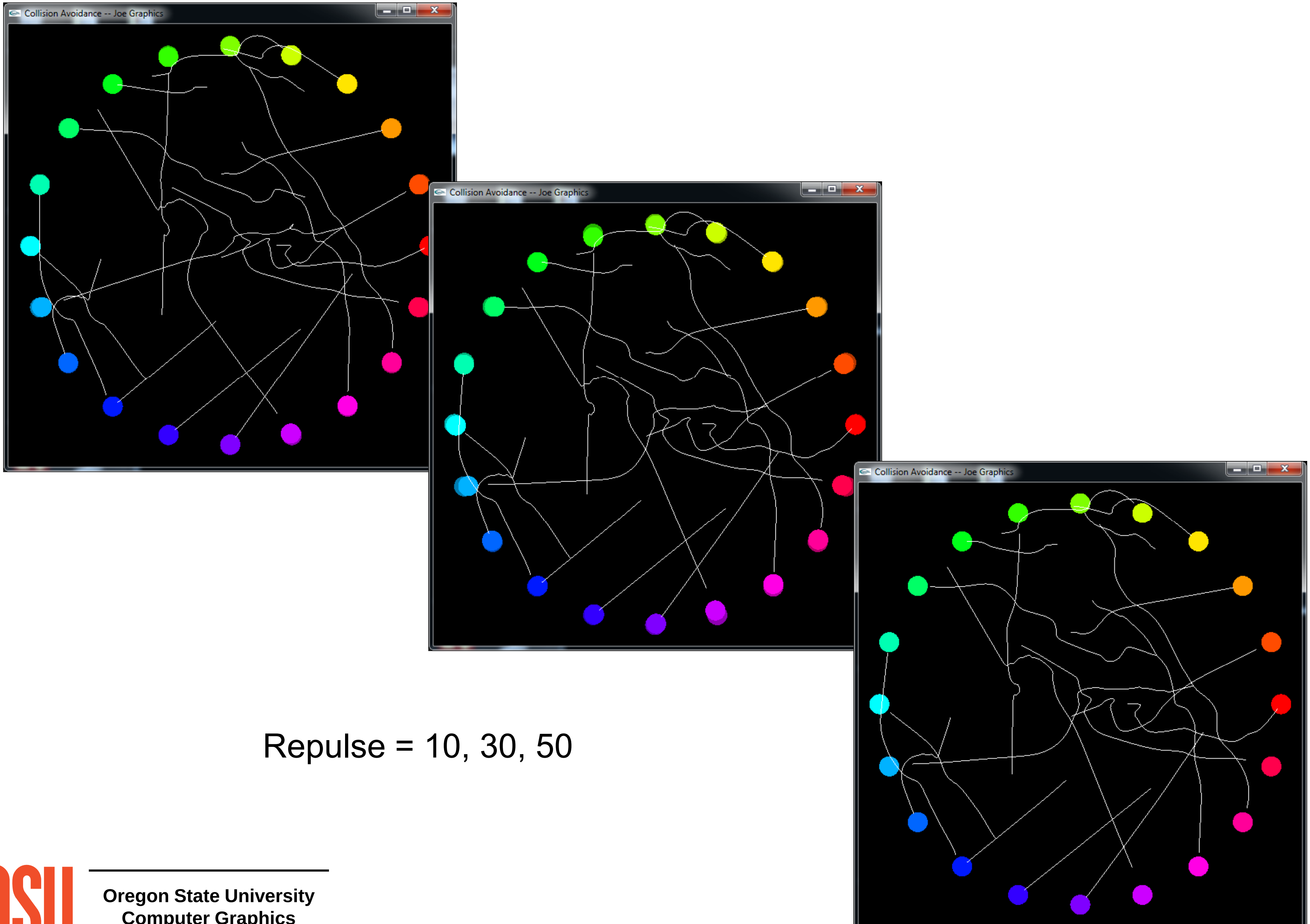
$$m\ddot{x} + c\dot{x} + kx = \sum F$$

Increasing the Stiffness



Stiffness = 3, 6, 9

Increasing the Repulsion Coefficient



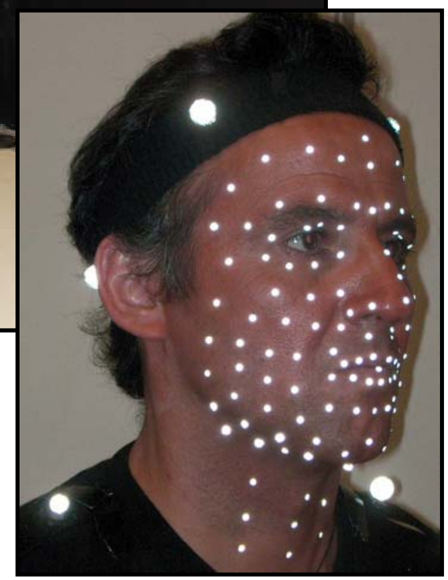
Repulse = 10, 30, 50

Motion Capture as an Input for Animation

NaturalPoint

Polhemus

Polhemus



MocapLab


Finding Additional
Information