Rendering Pipeline

Teacher: A.prof. Chengying Gao(高成英)

E-mail: mcsgcy@mail.sysu.edu.cn

School of Data and Computer Science

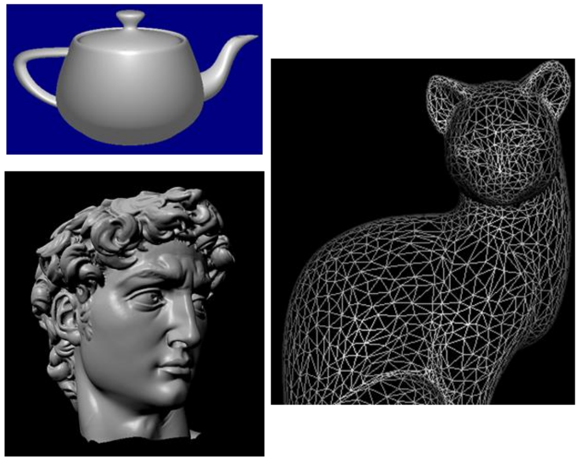
Outline

- **Computer Graphics System**
- Physical Imaging System
- Graphics Rendering Pipeline

Modeling
(Creating 3D Geometry)

→

Rendering
(Creating, shading images from geometry, lighting, materials)

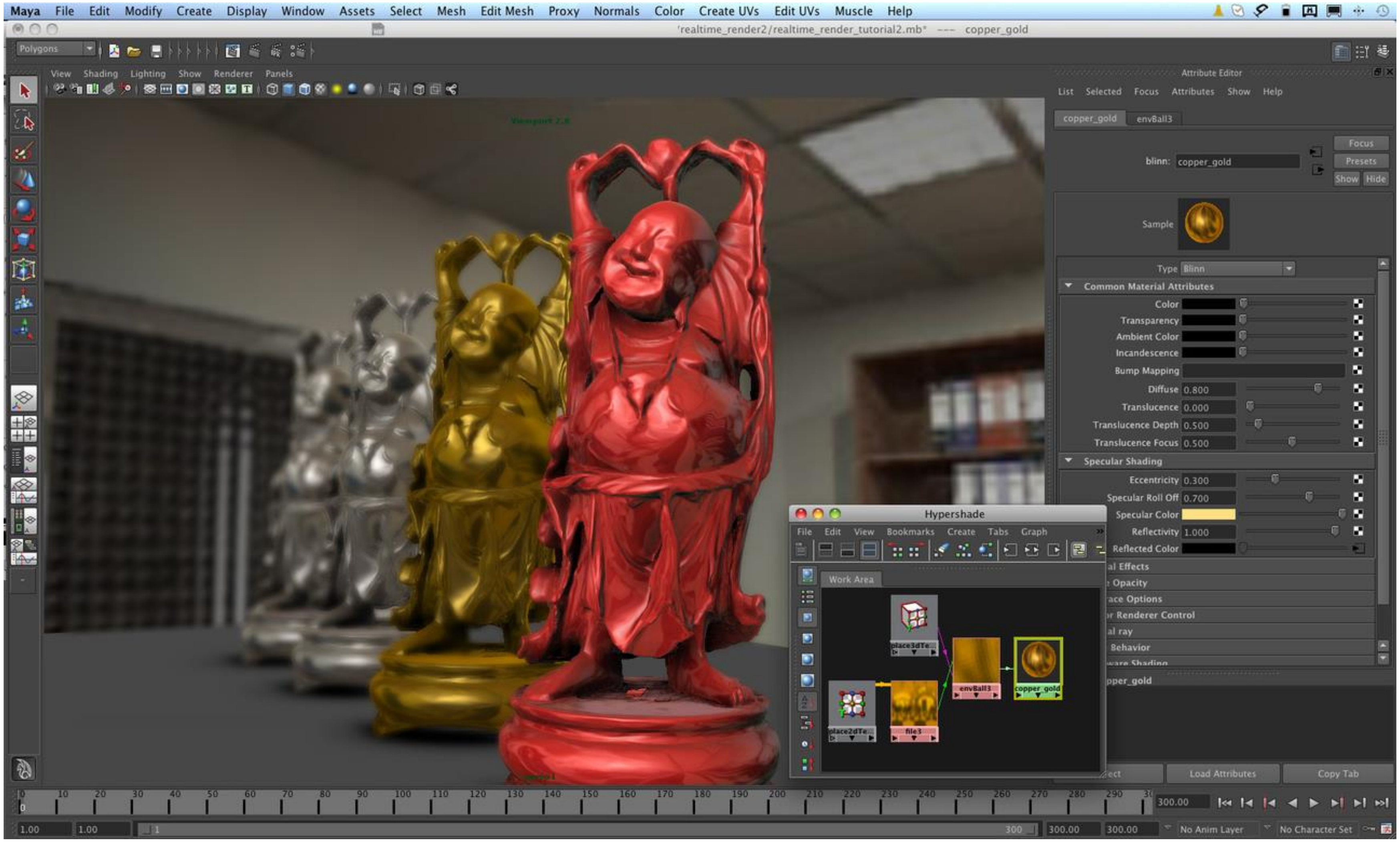


Example

- Where did these images come from?

- What hardware/software did we need to produce it?
  - Software: Maya for modeling and rendering but Maya is built on top of OpenGL
  - Hardware: PC with graphics card for modeling and rendering

Software – Autodesk Maya


Maya File Edit Modify Create Display Window Assets Select Mesh Edit Mesh Proxy Normals Color Create UVs Edit UVs Muscle Help
'realtime_render2/realtime_render_tutorial2.mb* --- copper_gold
Polygons
View Shading Lighting Show Renderer Panels
Attribute Editor
List Selected Focus Attributes Show Help
copper_gold envBall3
Focus
Presets
Show Hide
Sample
Common Material Attributes
Specular Shading
Hypershade
File Edit View Bookmarks Create Tabs Graph
Work Area
place3dTe
envBall3
copper_gold
place2dTe
file3
Load Attributes
Copy Tab
No Anim Layer
No Character Set

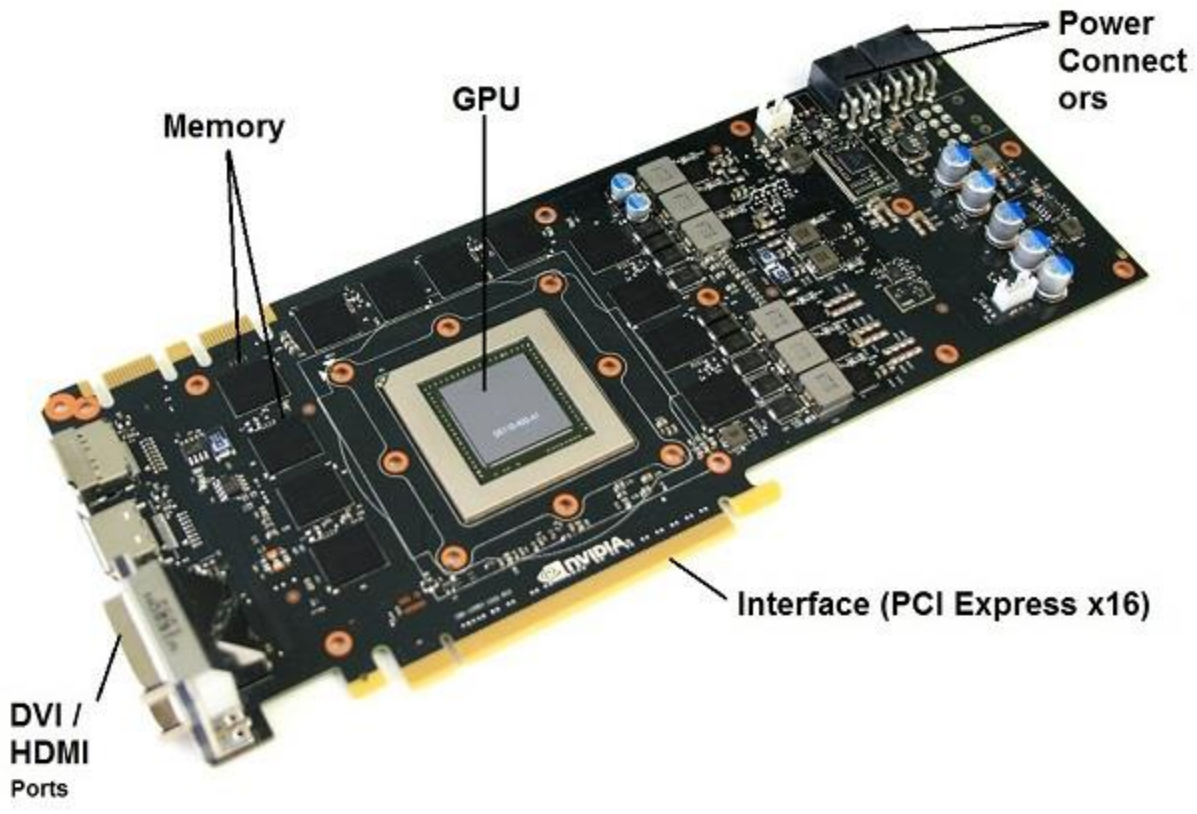
blinn: copper_gold

Type Blinn

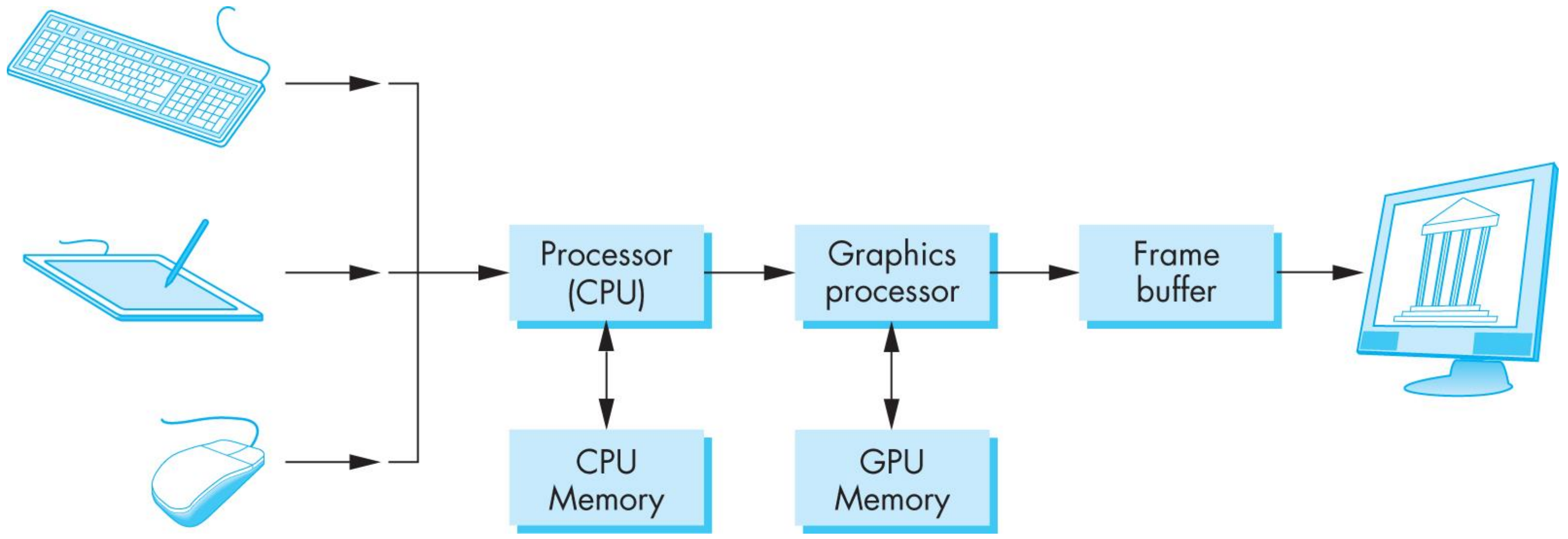
Diffuse 0.800
Translucence 0.000
Translucence Depth 0.500
Translucence Focus 0.500

Eccentricity 0.300
Specular Roll Off 0.700
Reflectivity 1.000

Hardware - display card



A computer graphics system



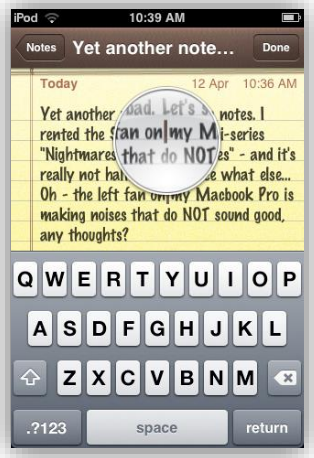
Input devices Image formed in frame buffer Output device

Angel and Shreiner: Interactive Computer Graphics 6E © Addison-Wesley 2012

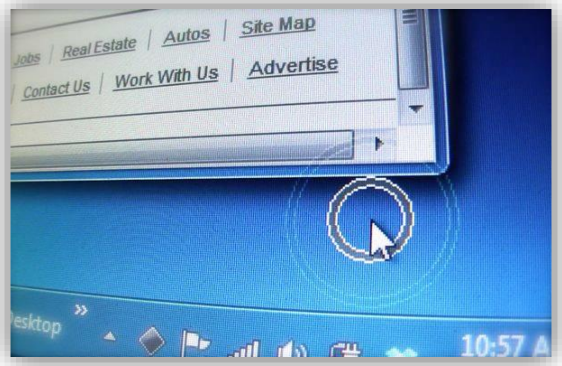
Input: Consumer devices

GRAPHICS

GRAPHICS



Yet another pad. Let's s notes. I rented the fan on my M-series "Nightmares that do NOT es" - and it's really not ha e what else... Oh - the left fan on my Macbook Pro is making noises that do NOT sound good, any thoughts?



Input: Image & video

The modern era of easy photography

Better photography in a simple way

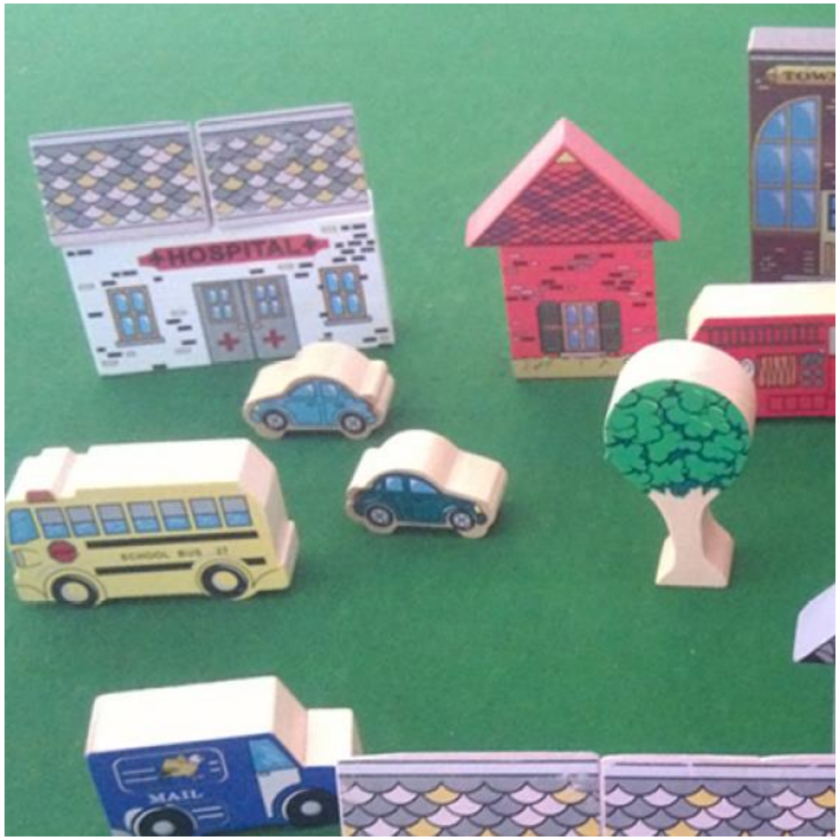
$ 5000

$ 5

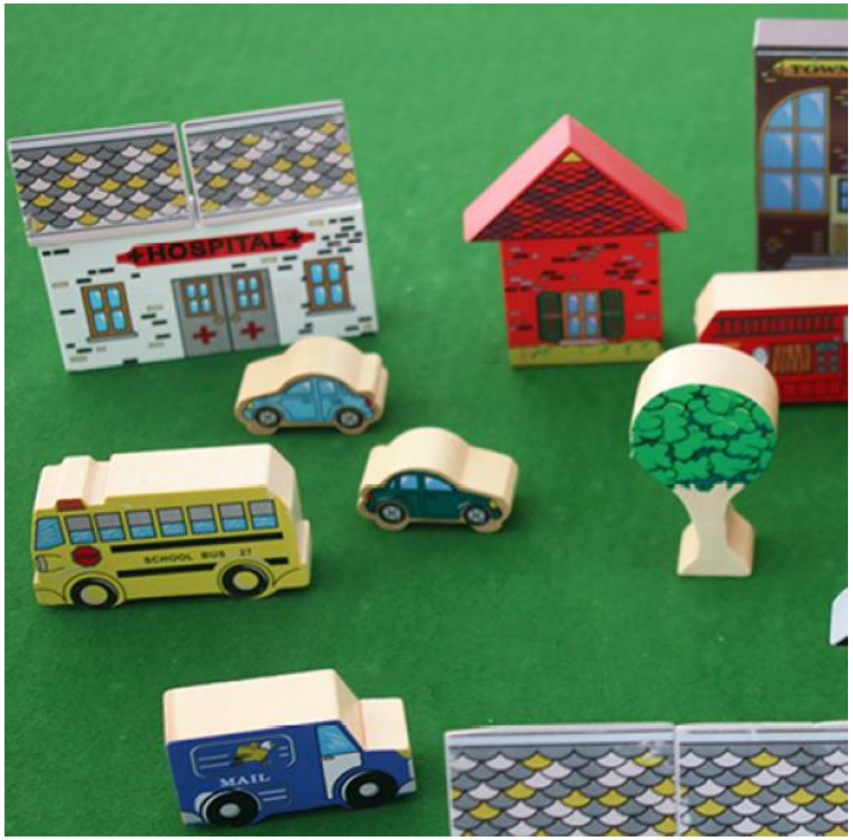
Mobile phone camera

Can these two ever be equally good at taking pictures?

Better photography in a simple way

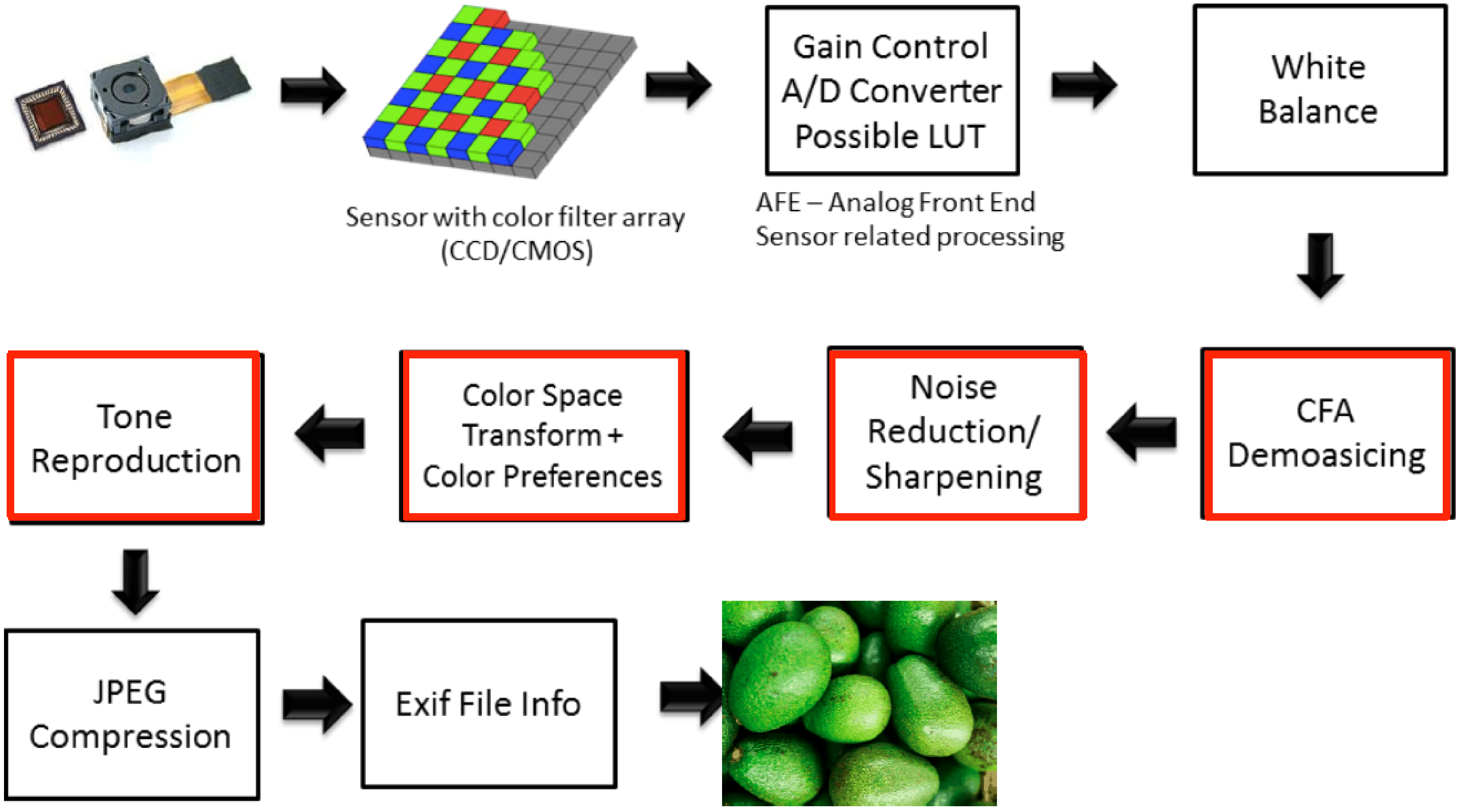


iPhone 3GS



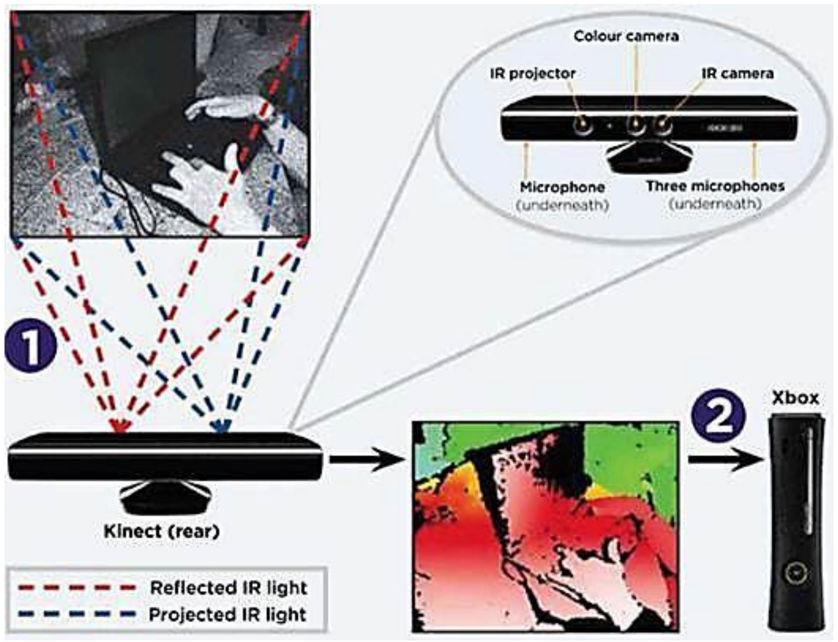
Canon EOS 5D Mark II

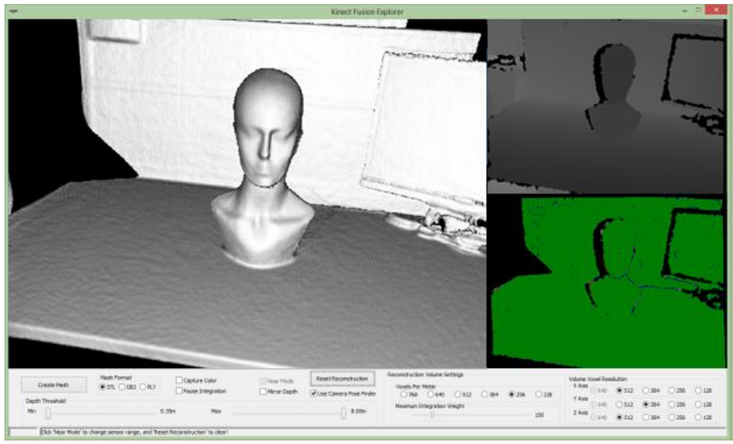
A standard model for imaging pipeline



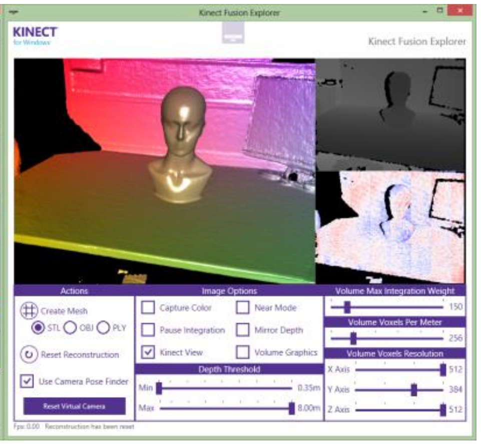
Input: Shape

Kinect Fusion



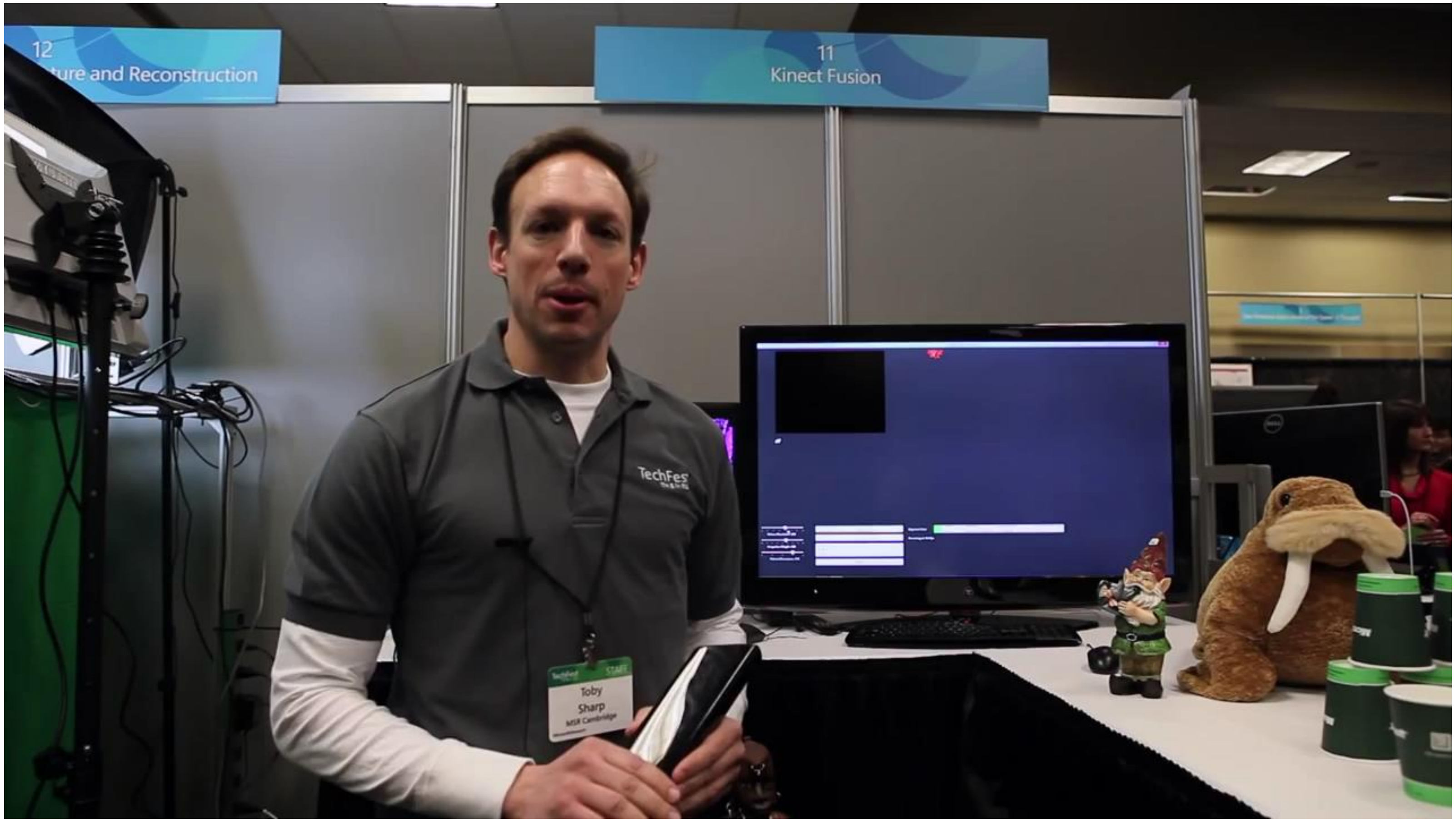


Click 'Near Mode' to change sensor range, and 'Reset Reconstruction' to clear!

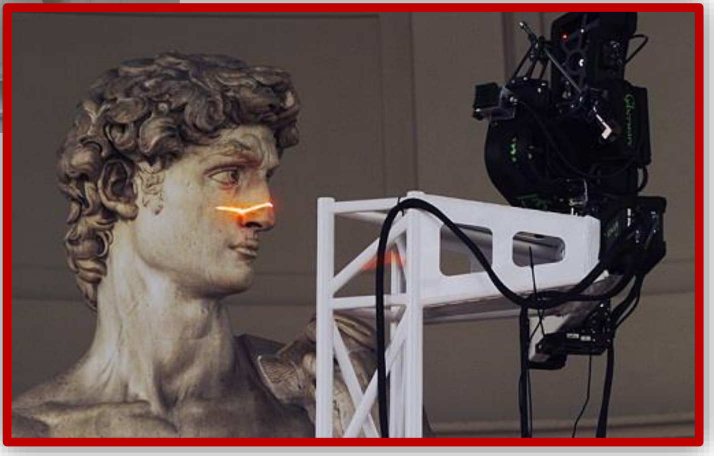


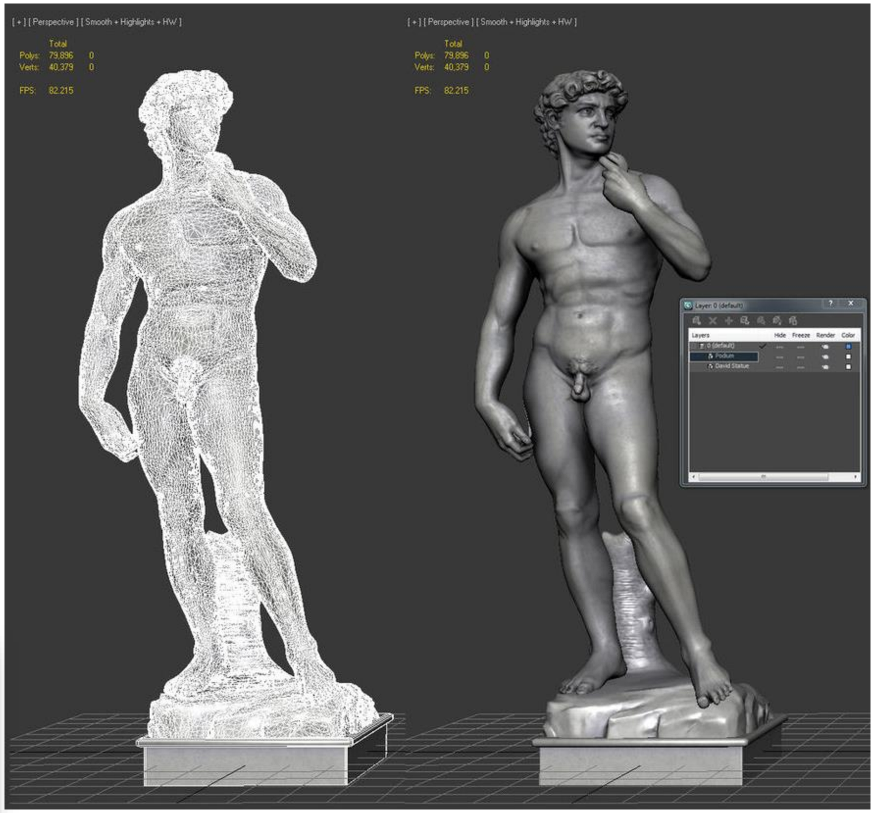
Fps: 0.00 Reconstruction has been reset

Shapes from Kinect Fusion



3D Scanning




[+] [Perspective] [Smooth + Highlights + HW]
Total
Polys: 79,896 0
Verts: 40,379 0
FPS: 82.215
[+] [Perspective] [Smooth + Highlights + HW]
Total
Polys: 79,896 0
Verts: 40,379 0
FPS: 82.215
Layer: 0 (default)
Layers
Hide
Freeze
Render
Color
0 (default)
Podium
David Statue

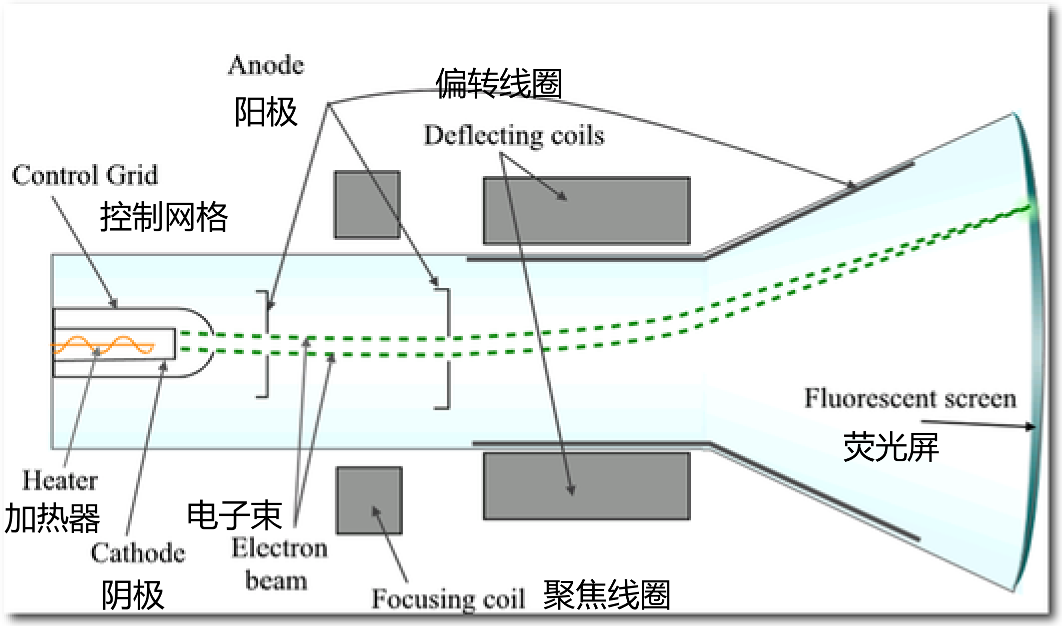
Desktop 3D scanner

Input: Specialized devices

What about the output device?

Display Devices

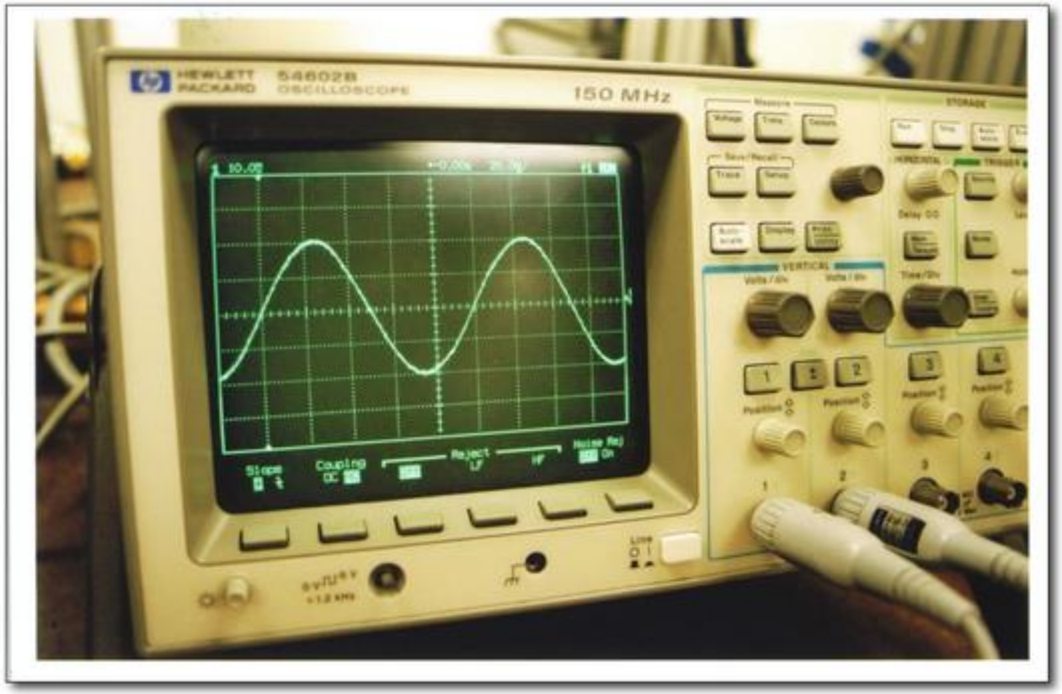
- Cathode Ray Tube (CRT)
  - CRT is a vacuum tube (电子管) containing one or more electron guns, and a phosphorescent screen (磷光屏) used to view images.



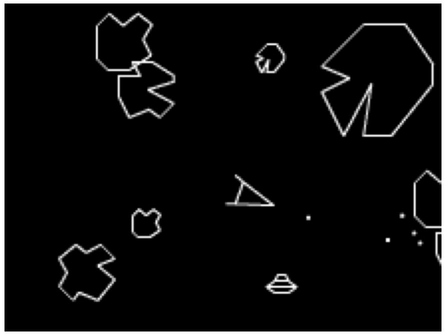
Karl Ferdinand Braun, 1897
Image courtesy of Wikipedia

Display Devices

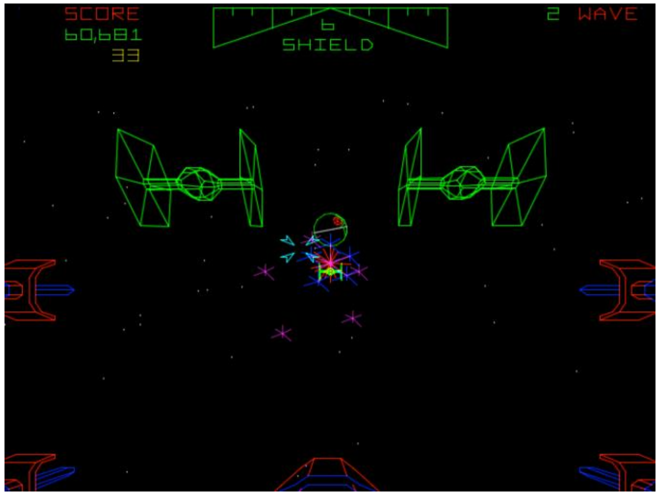
- Vector Displays



HP Oscilloscope



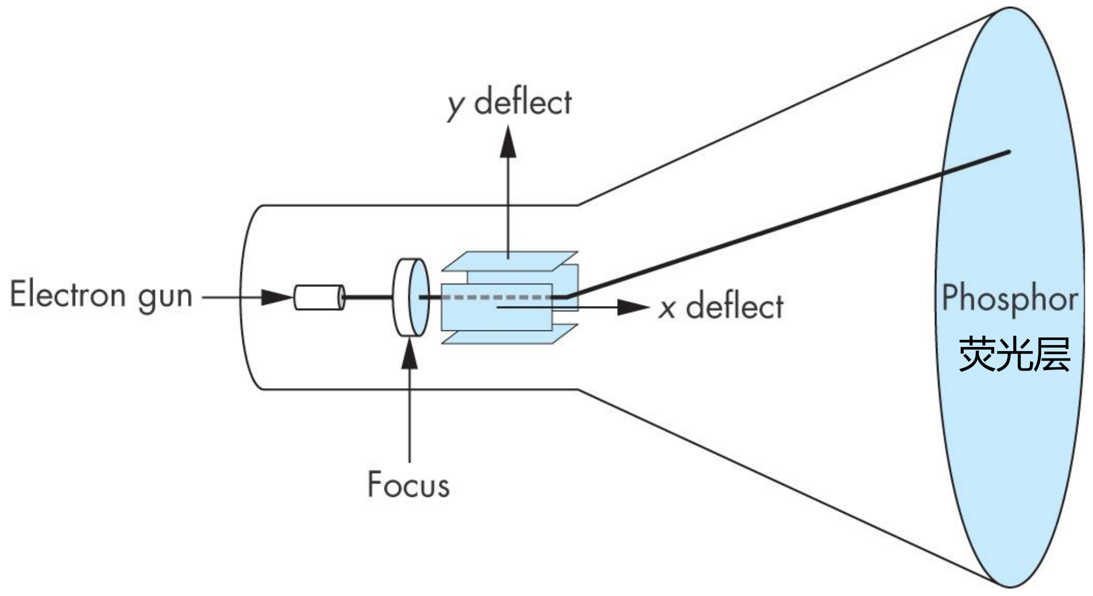
Asteroids, 1979 (行星游戏)



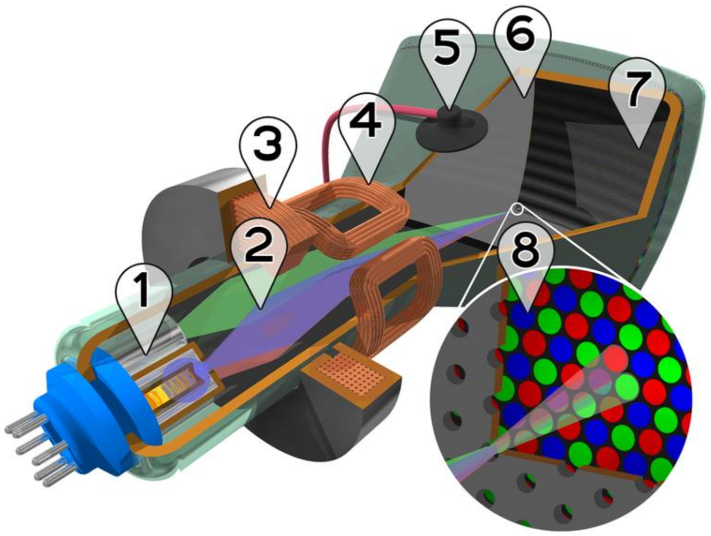
Star Wars, 1983 (星球大战)

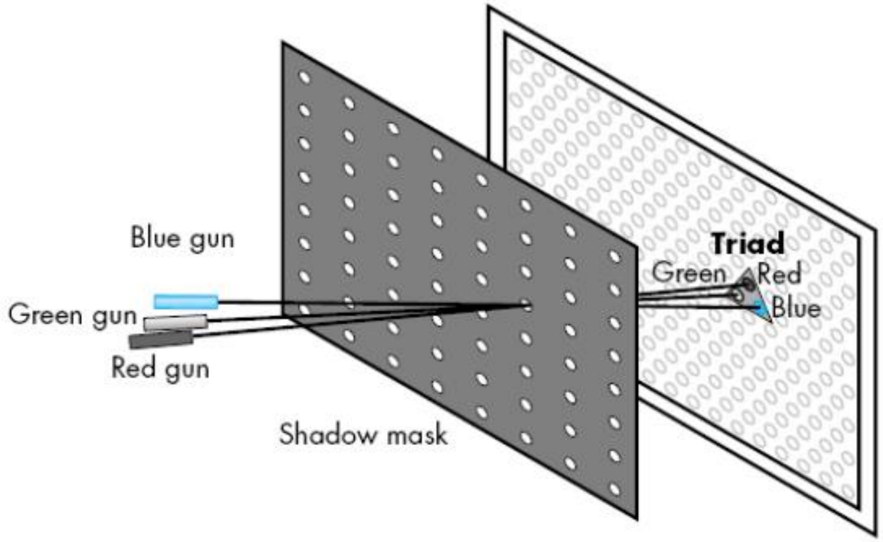
Display Devices

- Raster Display Devices (光栅显示器)
    - The refresh type raster scan display : Get the pixels from the frame buffer individually and corresponding location on the screen display.
    - Refresh Rate: 刷新率
    - Interlaced scan, Non-Interlaced or Progressive scan: 逐行扫描和隔行扫描



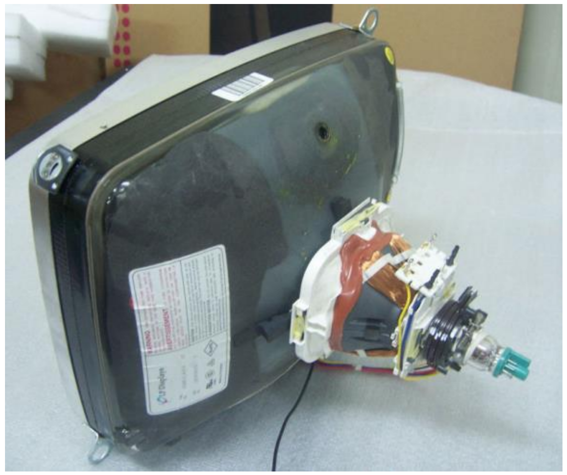
Color CRT display



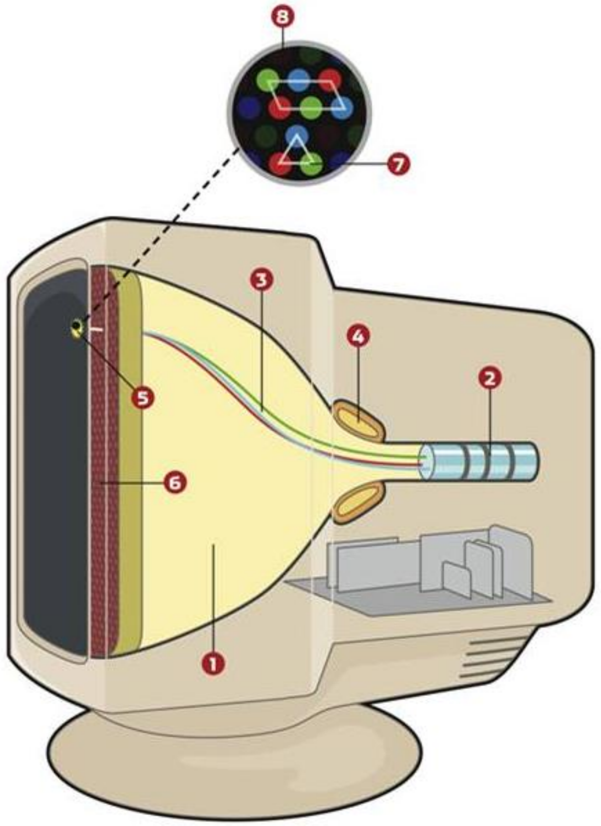


Cutaway rendering of a color CRT:

1. Three electron guns (for red, green, and blue phosphor dots)
2. Electron beams
3. Focusing coils
4. Deflection coils
5. Anode connection
6. Mask for separating beams for red, green, and blue part of displayed image
7. Phosphor layer with red, green, and blue zones
8. Close-up of the phosphor-coated inner side of the screen

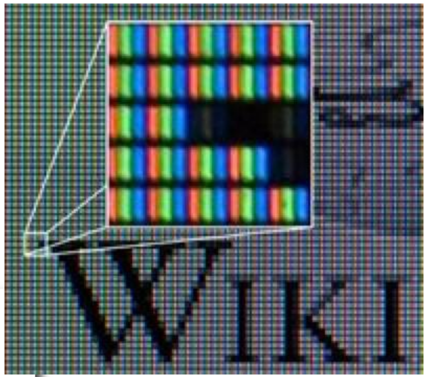


Color CRT display

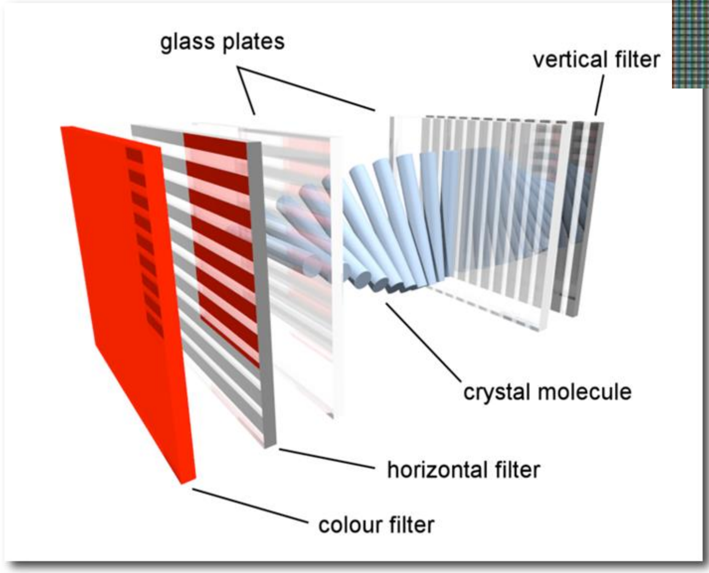


Display Devices

- **Flat-panel monitors**
  - LED (light-emitting diodes): 发光二级管
  - LCD (liquid-crystal displays): 液晶显示器
  - Plasma panels: 等离子显示器
- **Advantages**:
  - low consumption
  - small radiation
  - flicker free
  - No geometric distortion

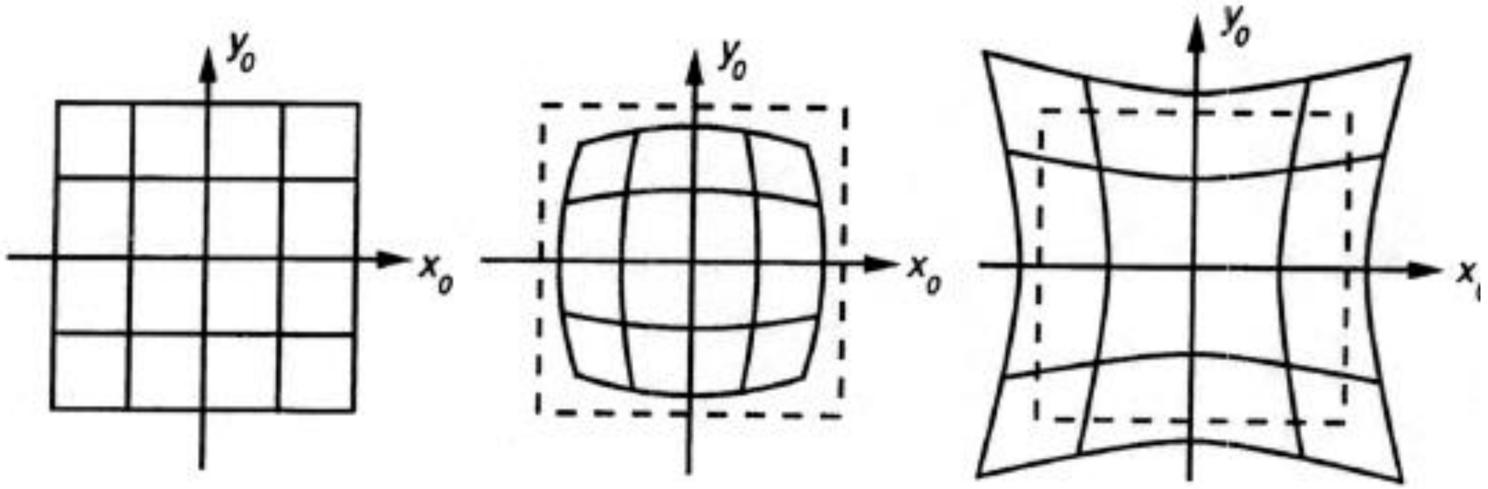


Close-up of pixels on an LCD display



A single subpixel of an LCD display

Geometric distortion in display monitor



No Distortion

无失真

Barrel Distortion

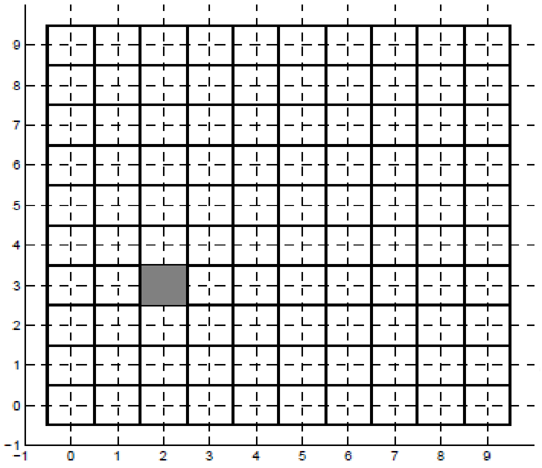
桶形失真

Pincushion Distortion

枕形失真

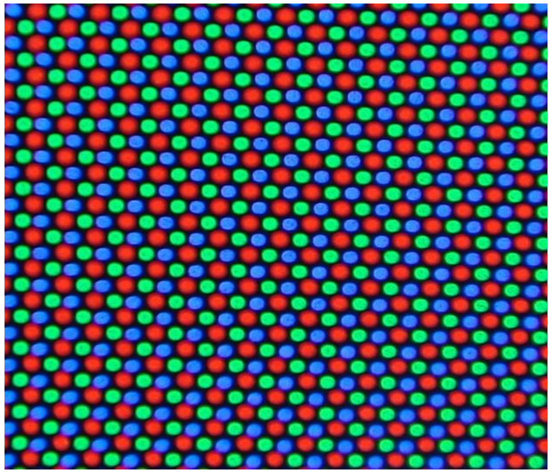
Modeling an image

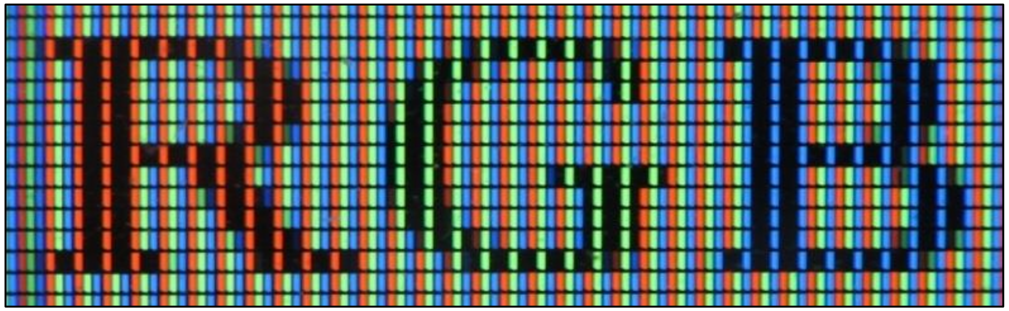
- Model a one-channel $\mathrm{m} \times \mathrm{n}$ image as the function $u(i, j)$
  - Maps pairs of integers (pixel coordinates) to real numbers
  - i and j are integers such that $0 \le i < \mathrm{m}$ and $0 \le j < \mathrm{n}$
- Associate each pixel value $u(i, j)$ to small area around display location with coordinates (i, j)
- A pixel here looks like a square centered over the sample point, but it's just a scalar value and the actual geometry of its screen appearance varies by device
  - Roughly circular spot on CRT
  - Rectangular on LCD panel



Pixels

- Pixels are point samples
- Point samples reconstructed for display (often using multiple subpixels for primary colors)


Close-up of a CRT screen


Close-up of an LCD screen

Frame buffer

- Frame buffer (帧缓冲区): a buffer that stores the contents of an image pixel by pixel
- Resolution (分辨率): the number of pixels per square inch on a computer-generated display
- Rasterization (光栅化)
- Graphics Processing Units (GPU, 图形处理器)

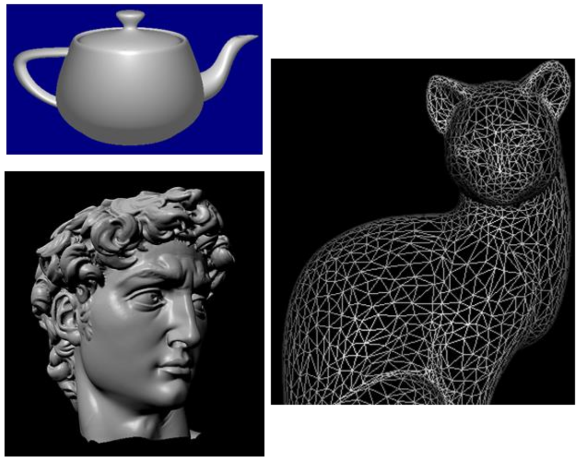
Outline

- Computer Graphics System
- **Physical Imaging System**
- Graphics Rendering Pipeline

Modeling
(Creating 3D Geometry)

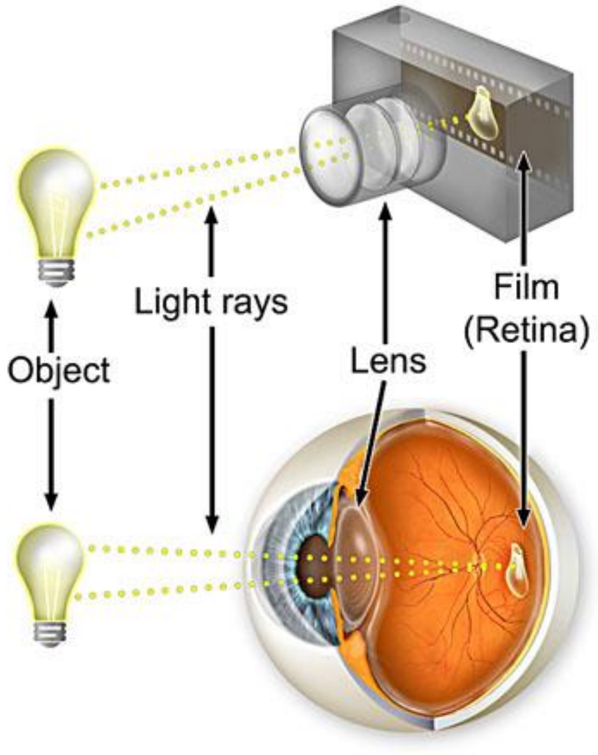
→

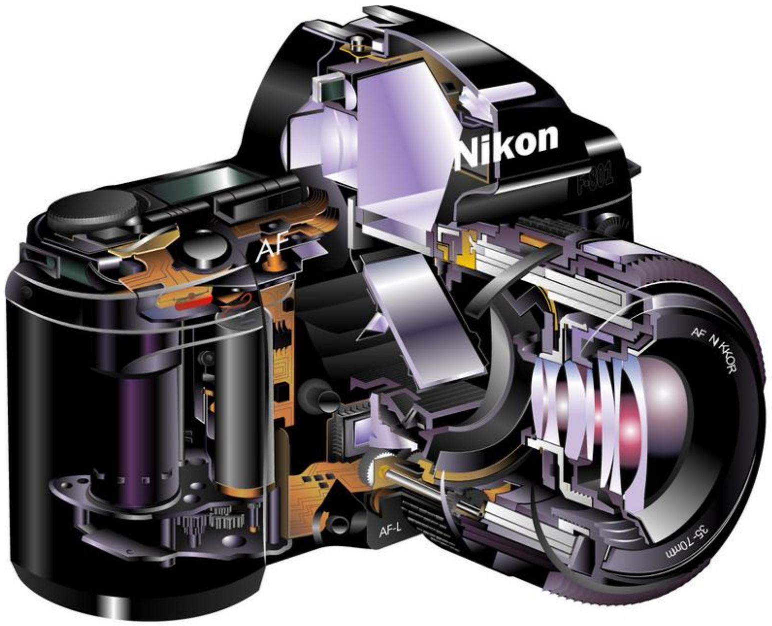
Rendering
(Creating, shading images from geometry, lighting, materials)



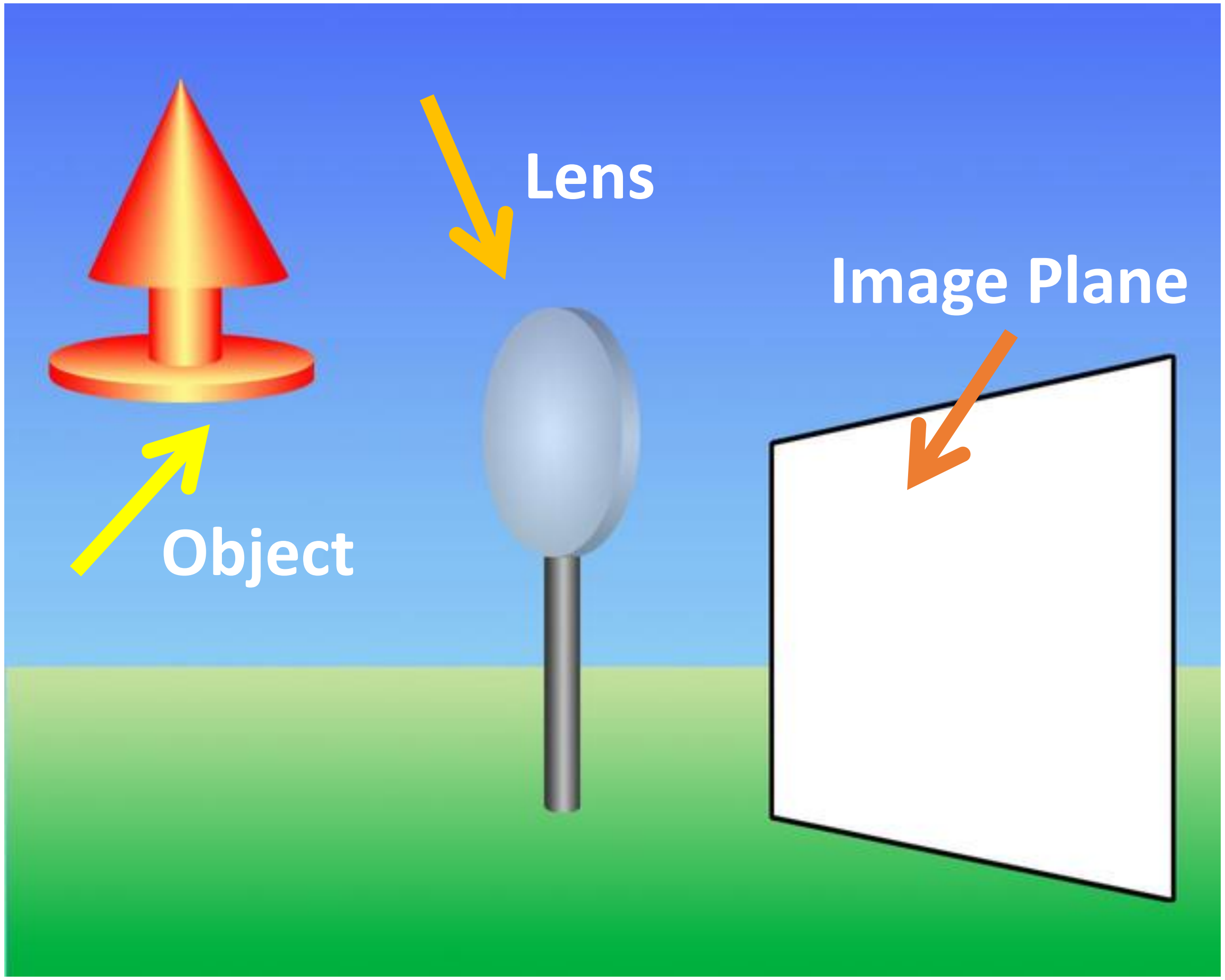
Physical Imaging System

- Eye(biology)
- Camera: film (chemistry),digital (physical + digital)

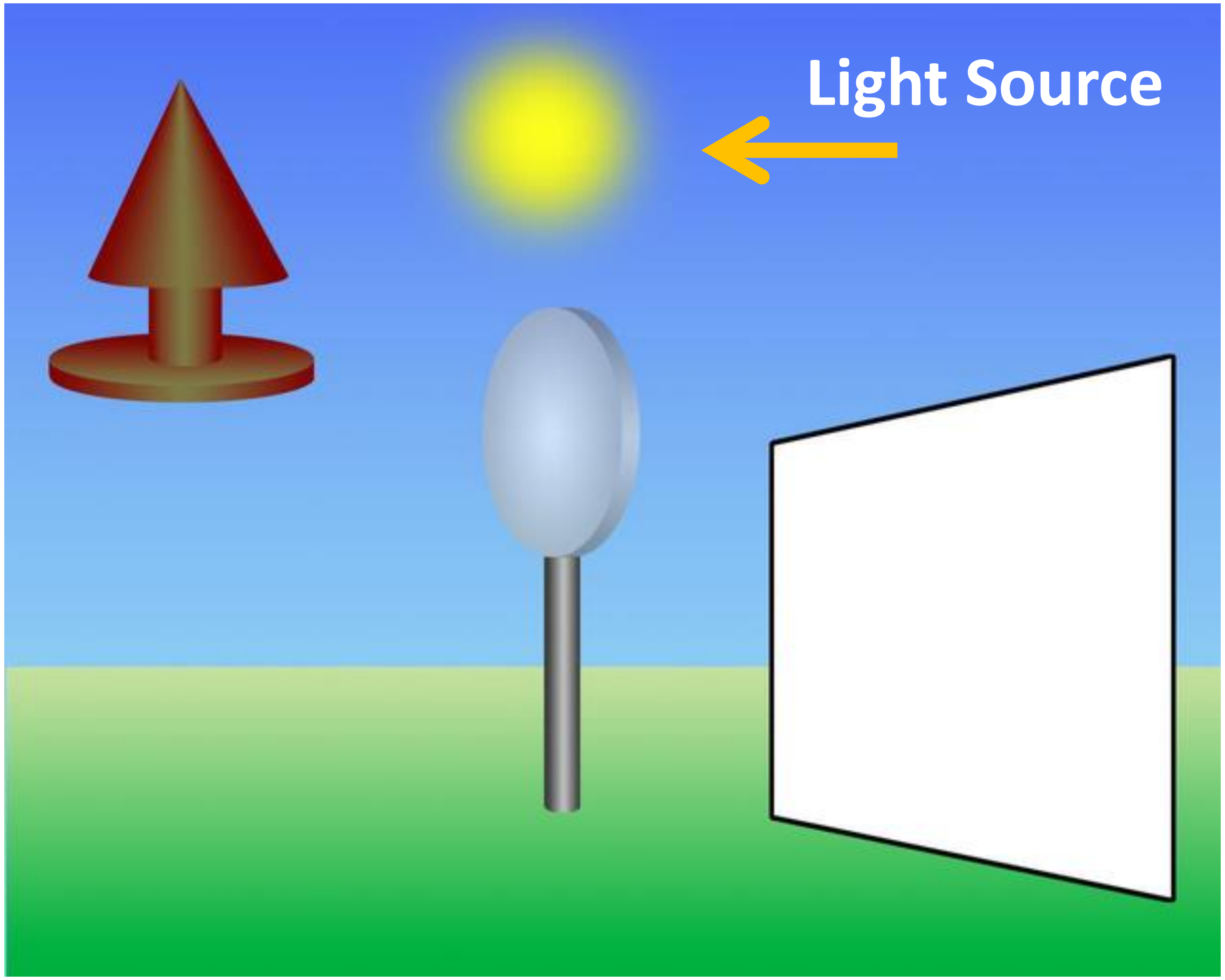




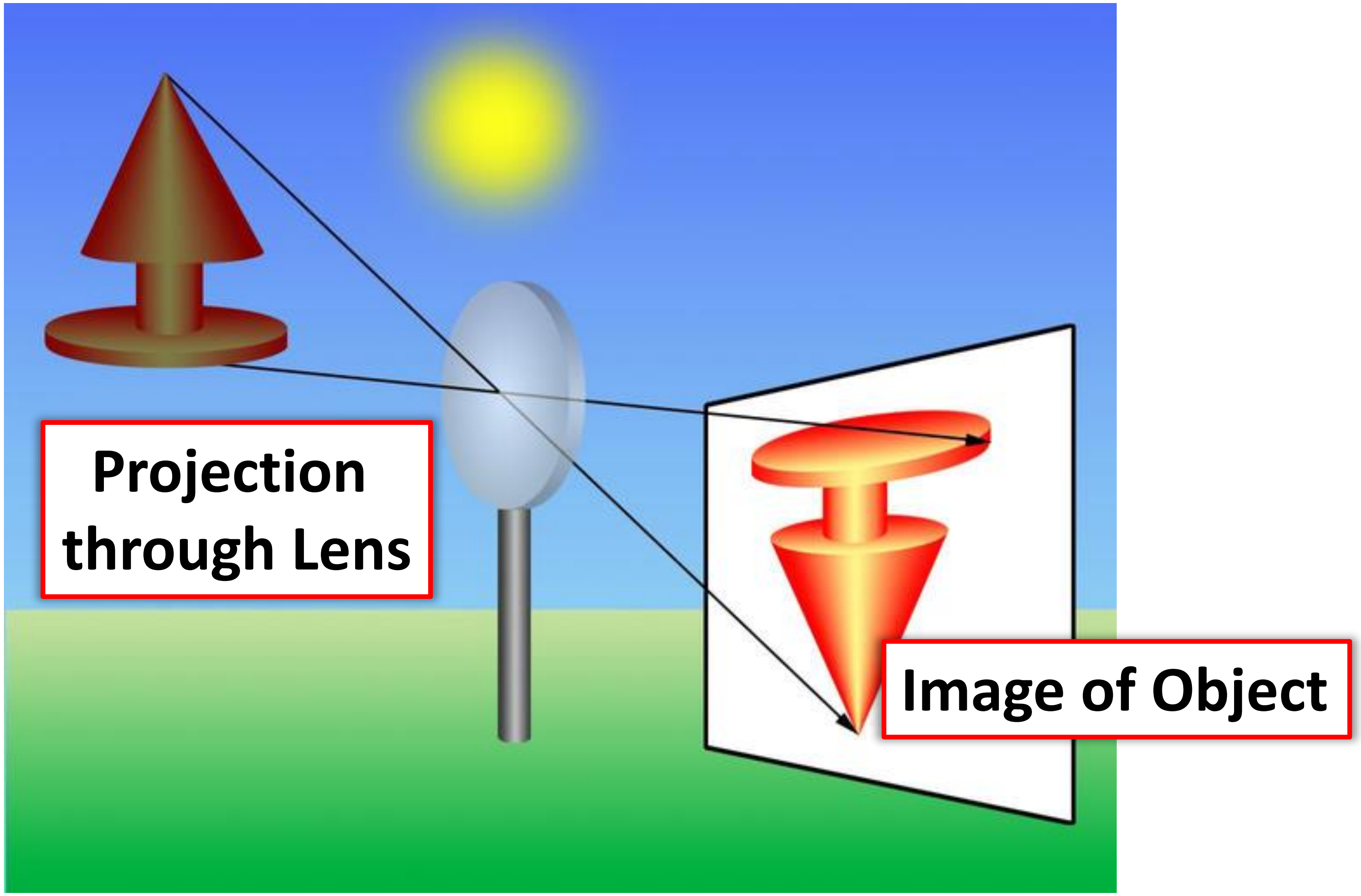
Imaging Principle



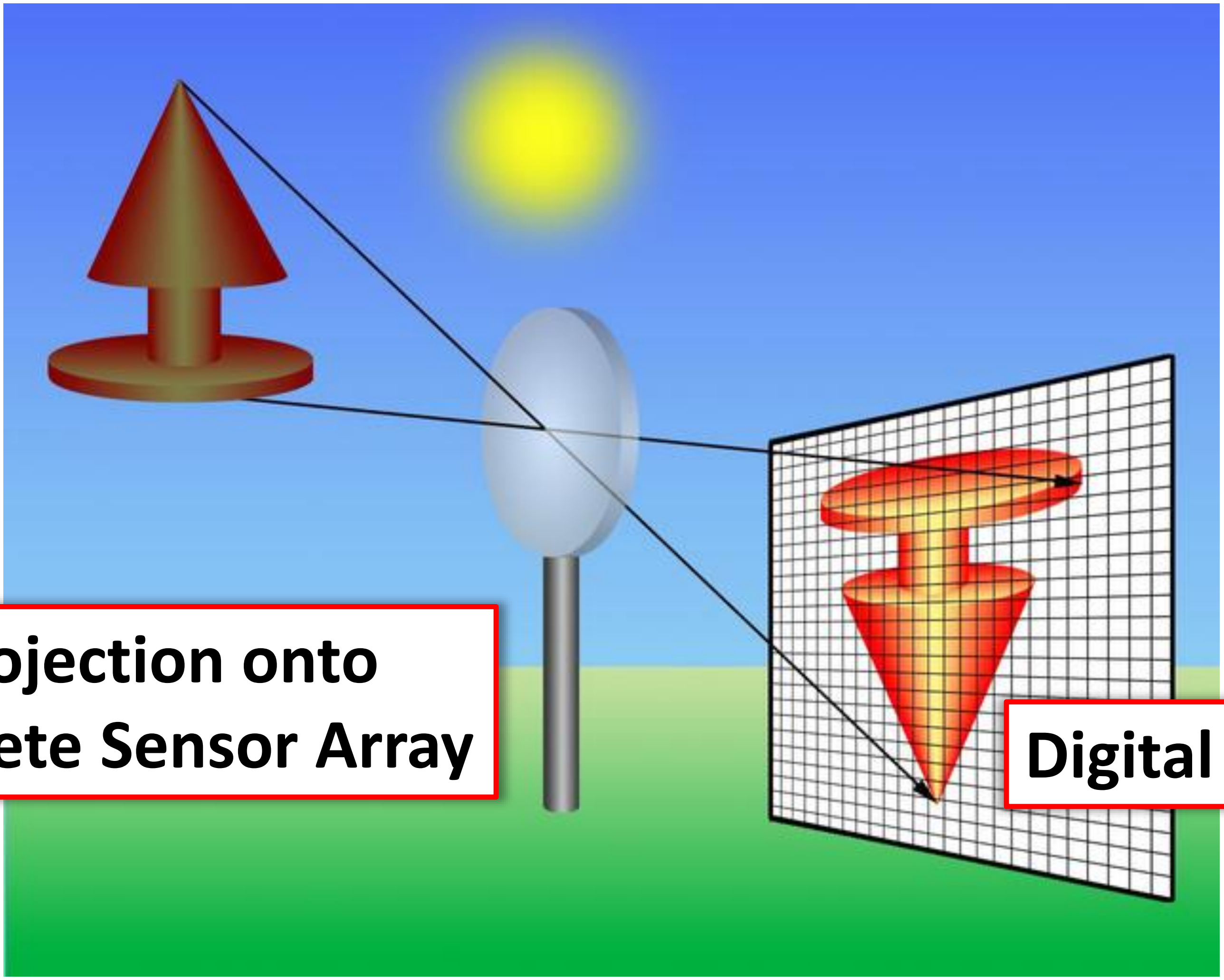
Imaging Principle



Imaging Principle



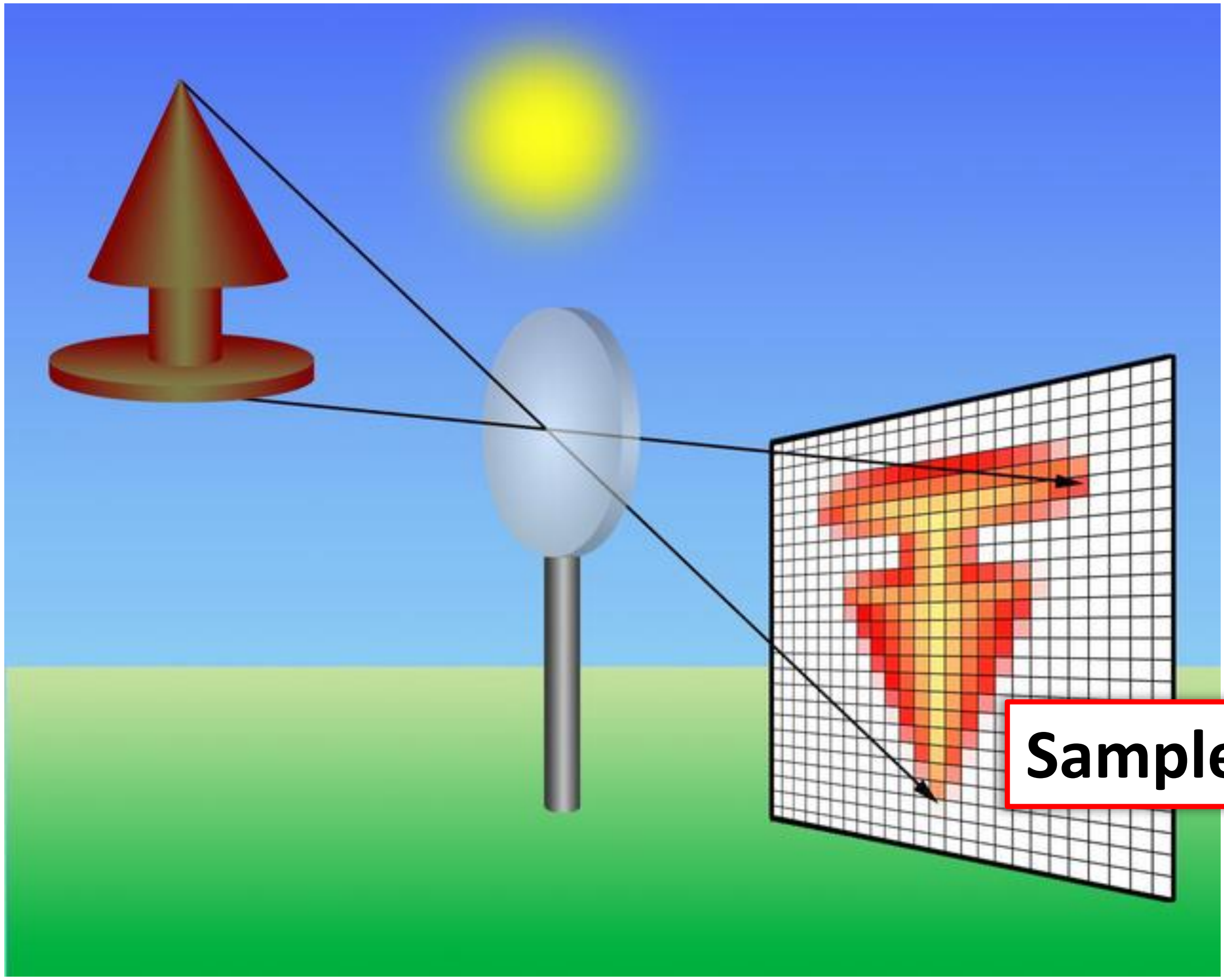
Imaging Principle



Projection onto
Discrete Sensor Array

Digital Camera

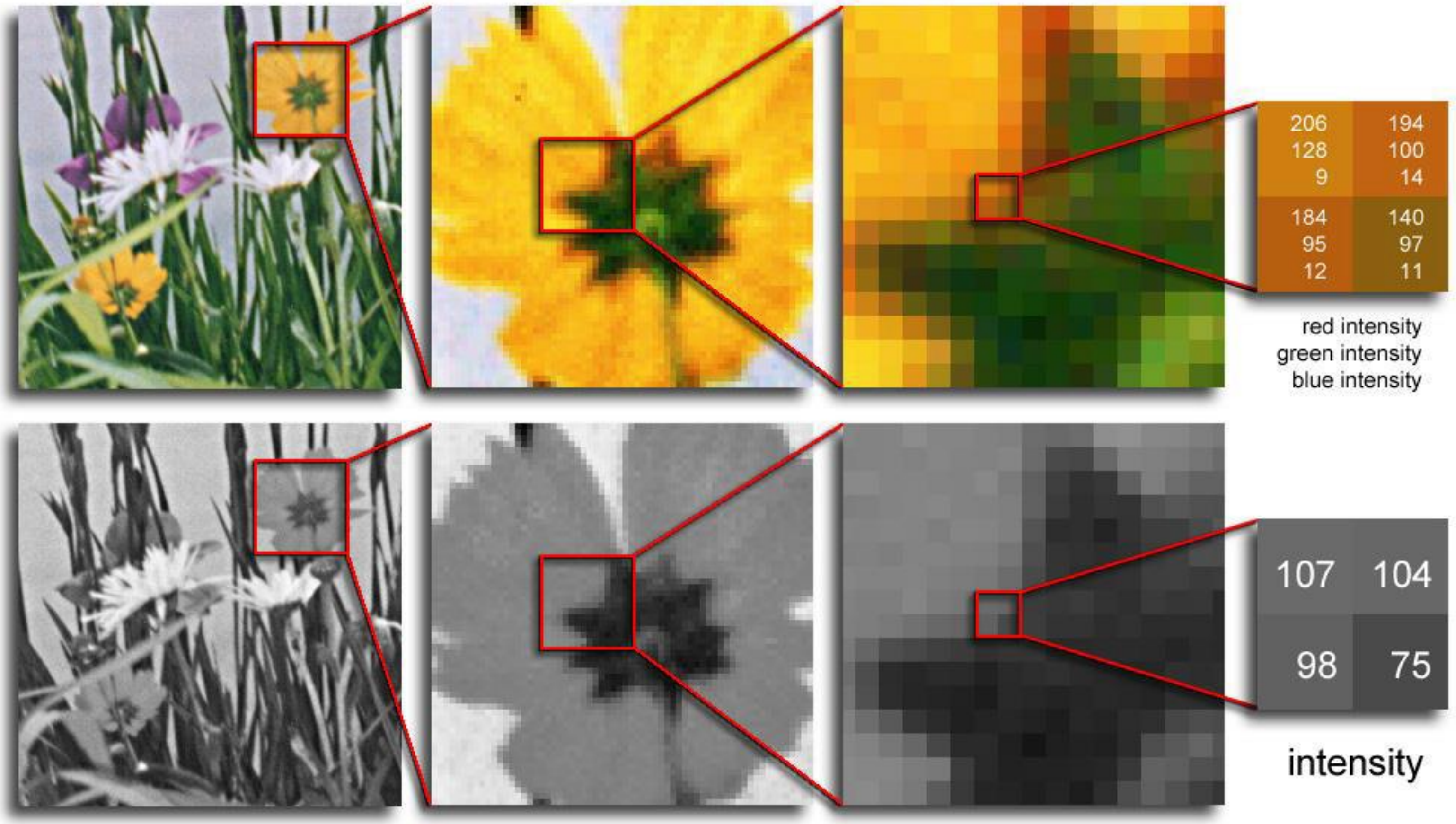
Imaging Principle


Sampled Image

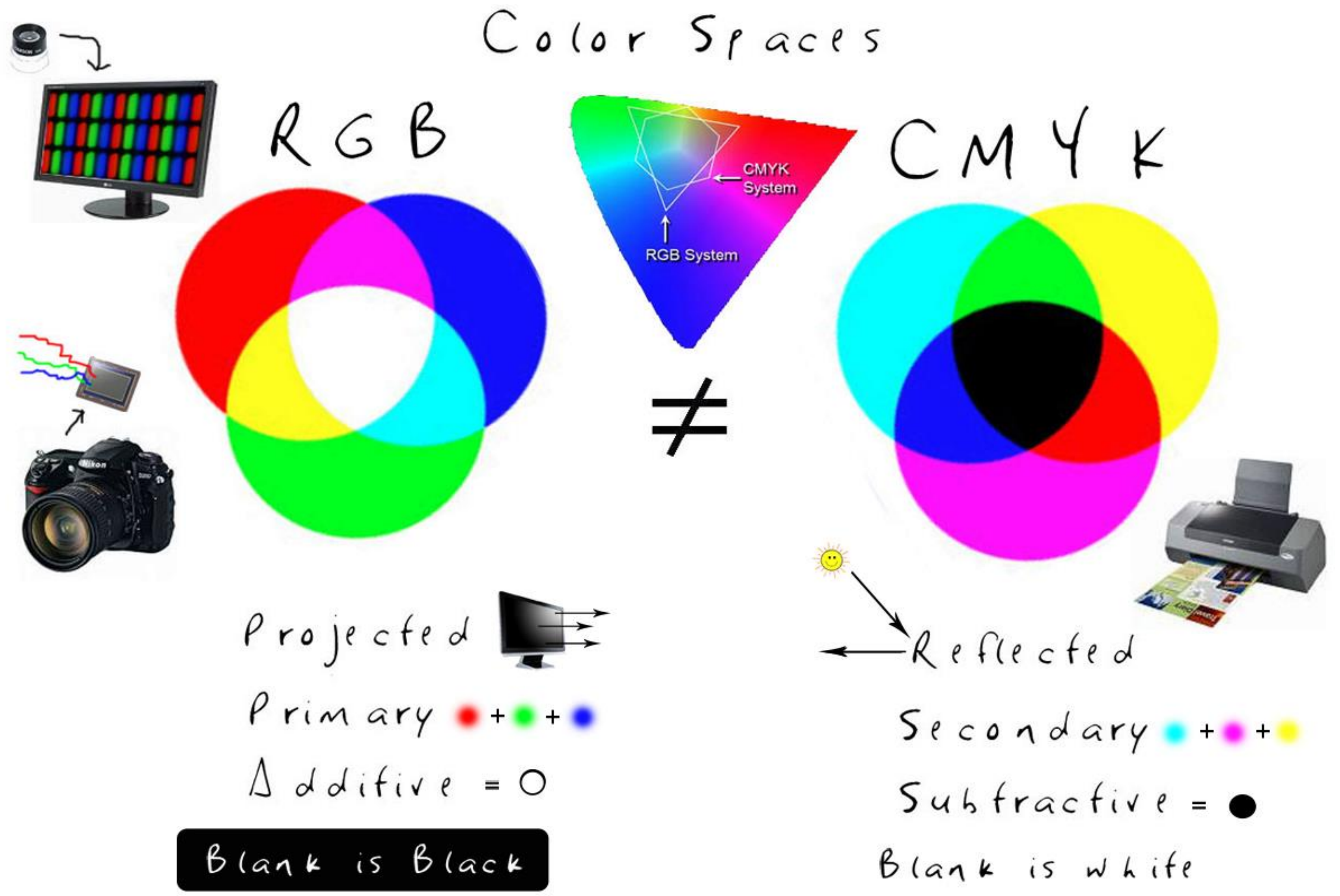
Digital Image

Color Image

Grey Image

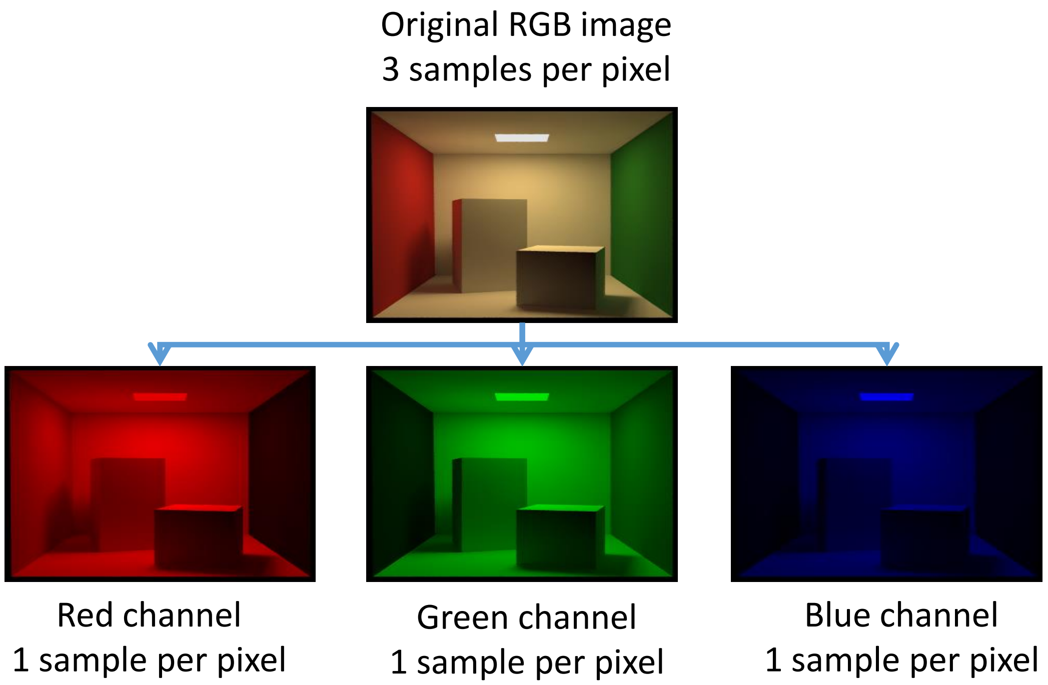


Color Space


Color Spaces
RGB
CMYK
CMYK System
RGB System
≠
Projected
Primary ● + ● + ●
Additive = ○
Blank is Black
Reflected
Secondary ● + ● + ●
Subtractive = ●
Blank is white

What is a channel?

- A channel is all the samples of a particular type
- RGB is a common format for image channels
  - Easy to implement in h/w
  - Corresponds approximately to human visual system anatomy (specialized "R, G, and B" cones)
  - Samples represent the intensity of the light at a point for a given wavelength (red, green, or blue)
- The R channel of an image is an image containing just the red samples



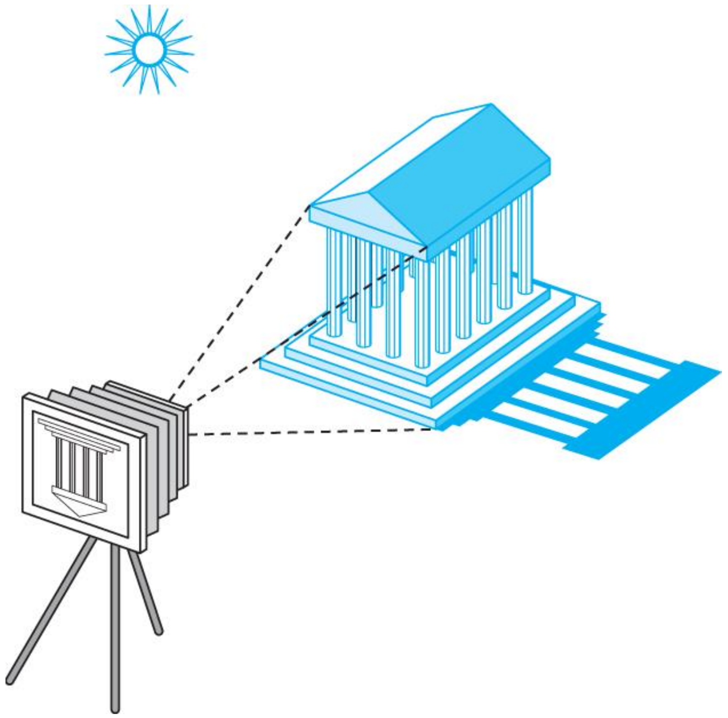
The alpha channel

- In addition to the R, G, and B channels of an image, add a fourth channel called α (transparency/opacity/translucency)
- Alpha varies between 0 and 1
  - Value of 1 represents a completely opaque pixel, one you cannot see through
  - Value of 0 is a completely transparent pixel
  - Value between $0 < \alpha < 1$ determines translucency
- Useful for blending images
  - Images with higher alpha values are less transparent
  - Linear interpolation ($\alpha X + (1 - \alpha)Y$) or full Porter-Duff compositing algebra

The orange box is drawn on top of the purple box using $\alpha = 0.8$

Image Synthetic Element

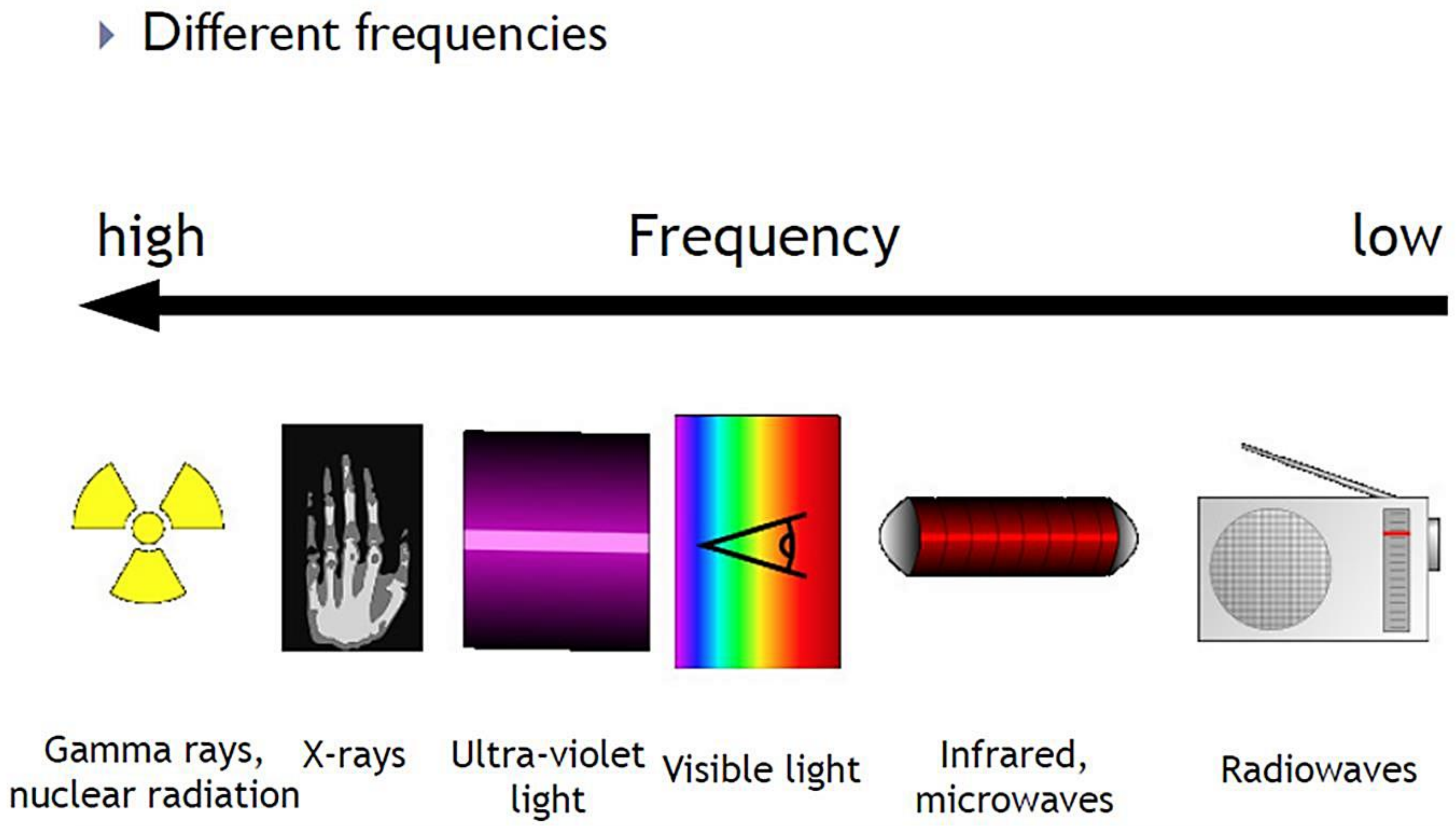
- Object
- Observer
- Light



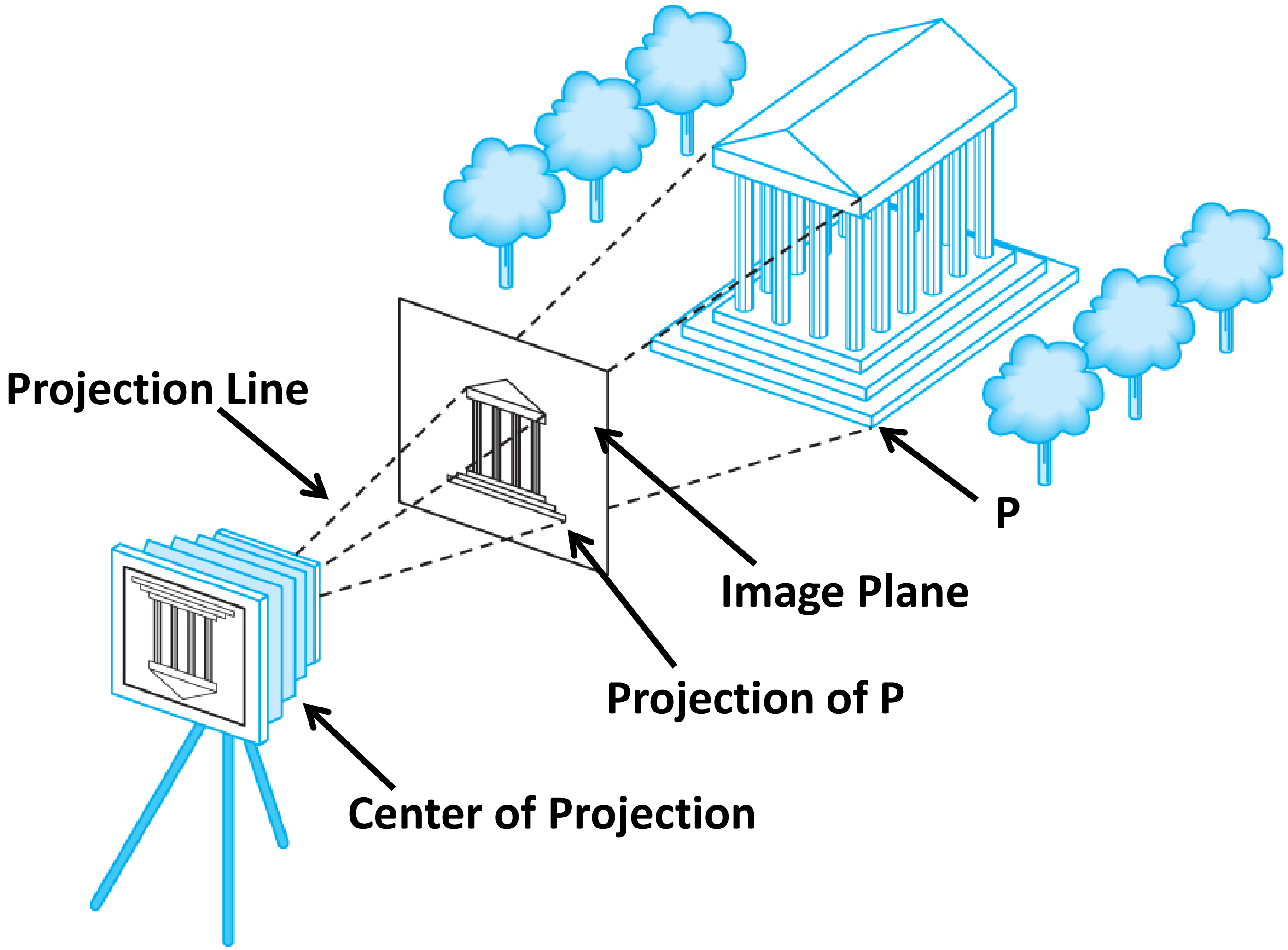
- Properties of Light, material: To determine the effect of light on the object

Light---Visible Light

- Light is a part of electromagnetic wave, it is a wave.
- It' visible range: between 350nm and 780nm



Synthetic camera model



Advantages of Synthetic camera model

- Object (物体), Observer (观察者) and light (光源) is complete independent
- 2D graphics is a special case of 3D
- Easy to implement by API
    - Set the properties of light, camera and material.
    - To determine the results of image by API.
- Quick hardware implement is supported :OpenGL, Direct 3D etc. is based on this model.

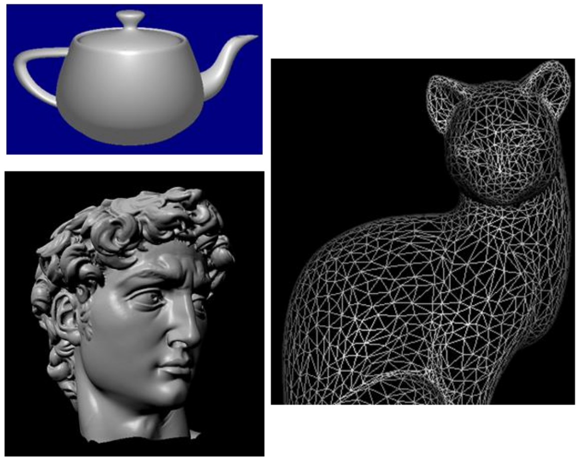
Outline

- Computer Graphics System
- Physical Imaging System
- **Graphics Rendering Pipeline**

Modeling
(Creating 3D Geometry)

→

Rendering
(Creating, shading images from geometry, lighting, materials)



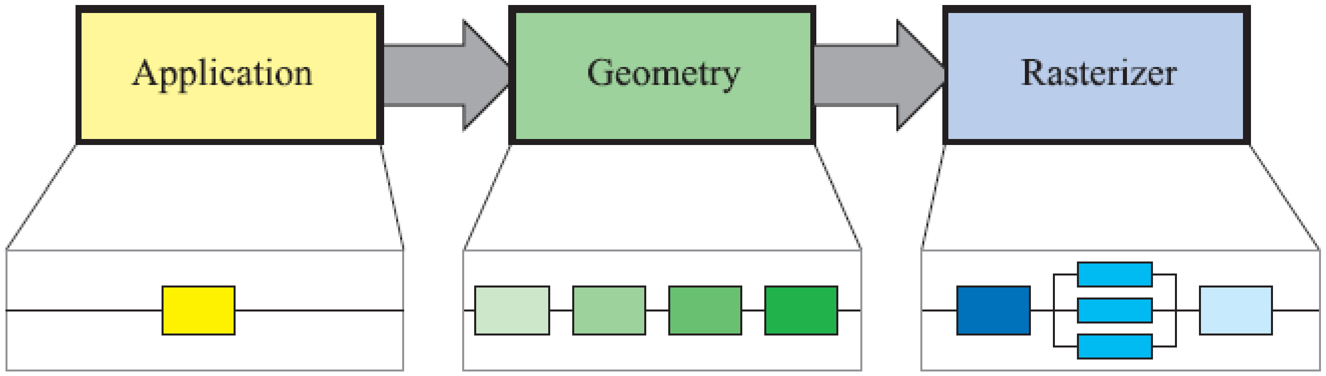
Demo - World War Z

Graphics Rendering Pipeline

- What is Rendering?
  - "Rendering is a process that takes as its input a set of objects and produces as its output an array of pixels."

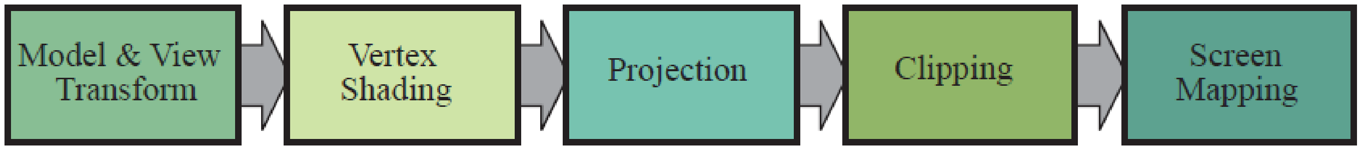
    -- 《Fundamentals of Computer Graphics (3rd Edition)》
- What is the rendering process (Graphic Pipeline)?
  - the sequence of steps used to create a 2D raster representation of a 3D scene.

Rendering Pipeline Stages

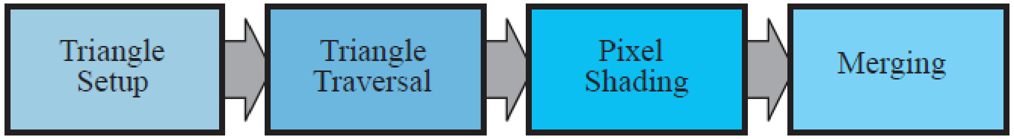


Rendering Pipeline Stages

- Application
  - e.g. collision detection, global acceleration algorithms, animation, physics simulation….. (On CPU)
- Geometry
  - Deal with transforms, projection. Computes what is to be draw, how it should be drawn, and where it should be drawn (On GPU)



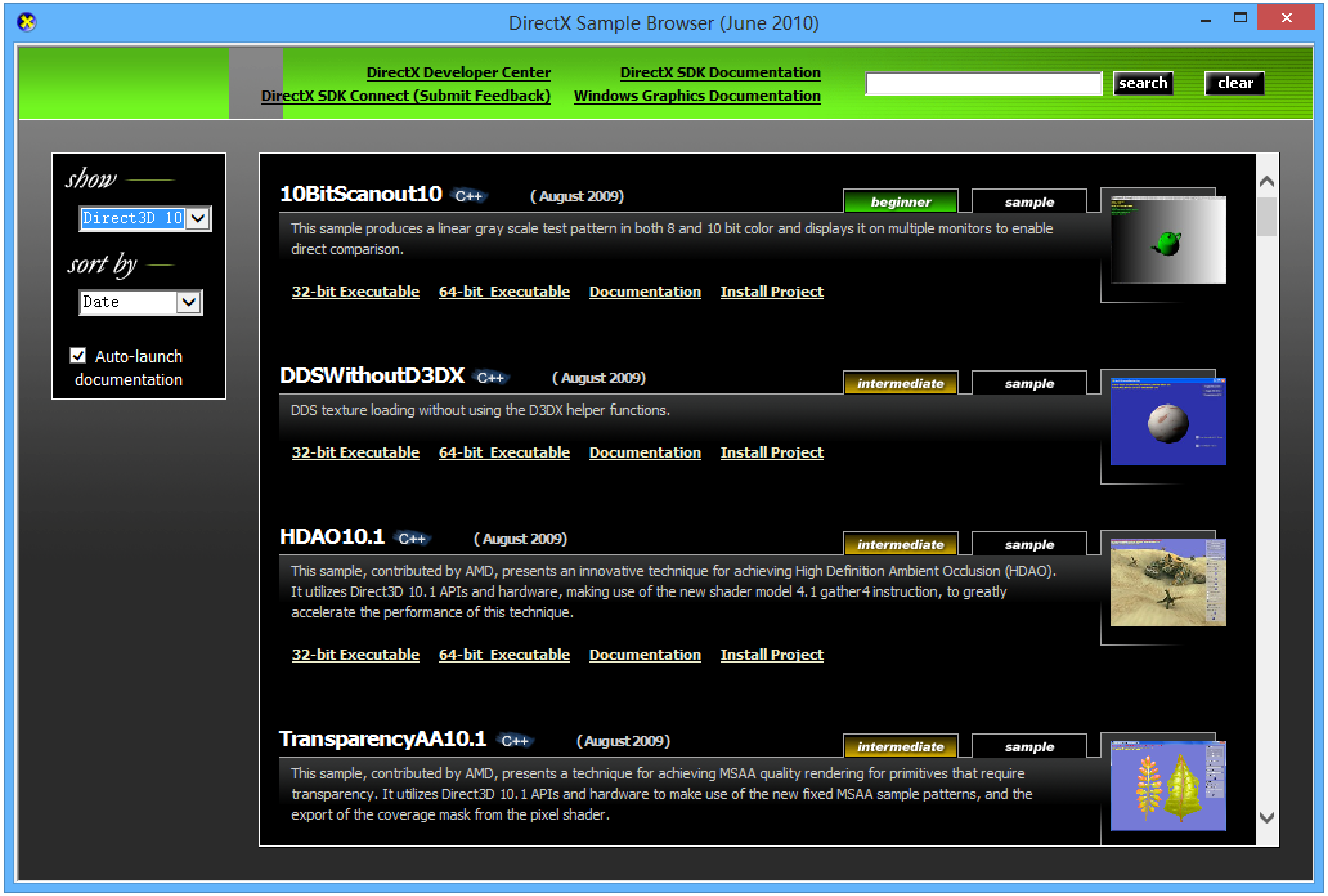
- Rasterizer
  - Conversion from two-dimensional vertices in screen space – each with a z-value (depth value), and various shading information associated with each vertex – into pixels on the screen (On GPU)



Rendering Pipeline

- Abstract model
  - sequence of operations to transform geometric model into digital image
  - graphics hardware workflow
- Underlying API model for programming graphics hardware
  - OpenGL
  - Direct 3D
- Many actual implementations

Direct 3D Samples



DirectX Sample Browser (June 2010)

DirectX Developer Center DirectX SDK Documentation
DirectX SDK Connect (Submit Feedback) Windows Graphics Documentation

10BitScanout10 C++ (August 2009) beginner sample

This sample produces a linear gray scale test pattern in both 8 and 10 bit color and displays it on multiple monitors to enable direct comparison.

32-bit Executable 64-bit Executable Documentation Install Project

DDSWithoutD3DX C++ (August 2009) intermediate sample

DDS texture loading without using the D3DX helper functions.

32-bit Executable 64-bit Executable Documentation Install Project

HDAO10.1 C++ (August 2009) intermediate sample

This sample, contributed by AMD, presents an innovative technique for achieving High Definition Ambient Occlusion (HDAO). It utilizes Direct3D 10.1 APIs and hardware, making use of the new shader model 4.1 gather4 instruction, to greatly accelerate the performance of this technique.

32-bit Executable 64-bit Executable Documentation Install Project

TransparencyAA10.1 C++ (August 2009) intermediate sample

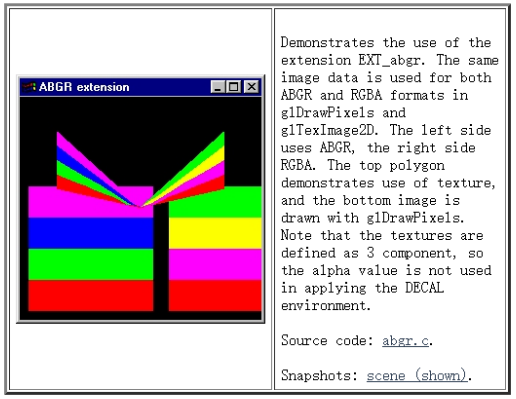
This sample, contributed by AMD, presents a technique for achieving MSAA quality rendering for primitives that require transparency. It utilizes Direct3D 10.1 APIs and hardware to make use of the new fixed MSAA sample patterns, and the export of the coverage mask from the pixel shader.

OpenGL Samples

- GLUT demos

https://www.opengl.org/archives/resources/code/samples/glut_examples/examples/examples.html

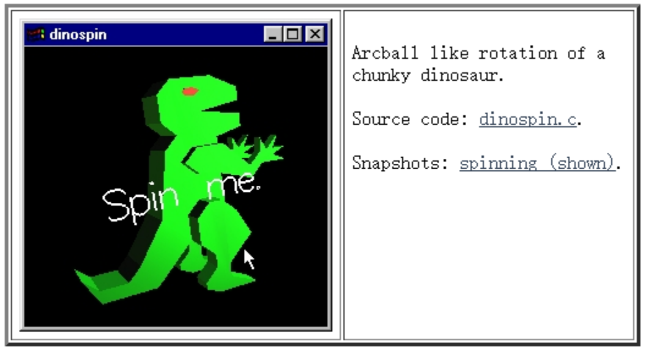
examples.zip



Demonstrates the use of the extension EXT_abgr. The same image data is used for both ABGR and RGBA formats in glDrawPixels and glTexImage2D. The left side uses ABGR, the right side RGBA. The top polygon demonstrates use of texture, and the bottom image is drawn with glDrawPixels. Note that the textures are defined as 3 component, so the alpha value is not used in applying the DECAL environment.

Source code: abgr.c.

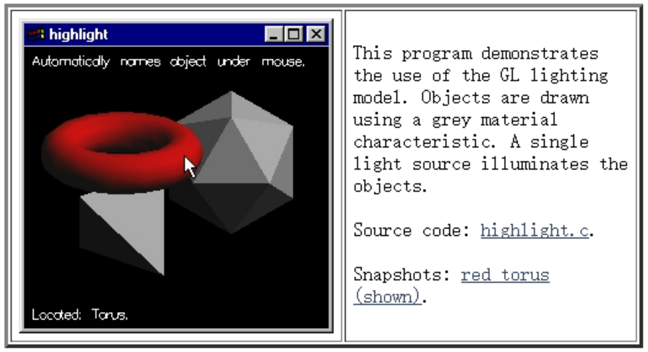
Snapshots: scene (shown).



Arcball like rotation of a chunky dinosaur.

Source code: dinospin.c.

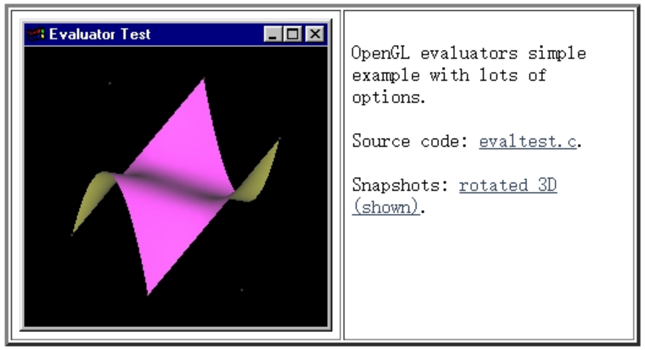
Snapshots: spinning (shown).



This program demonstrates the use of the GL lighting model. Objects are drawn using a grey material characteristic. A single light source illuminates the objects.

Source code: highlight.c.

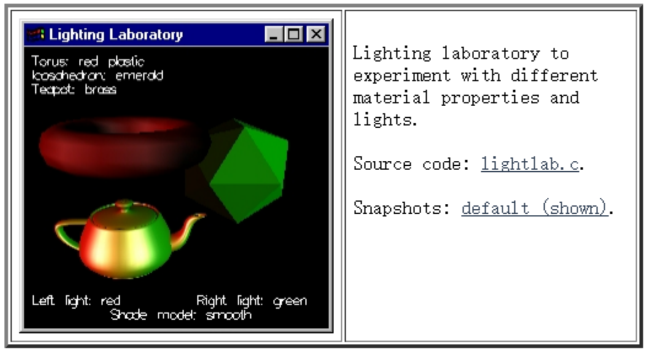
Snapshots: red torus (shown).



OpenGL evaluators simple example with lots of options.

Source code: evaltest.c.

Snapshots: rotated 3D (shown).



Lighting laboratory to experiment with different material properties and lights.

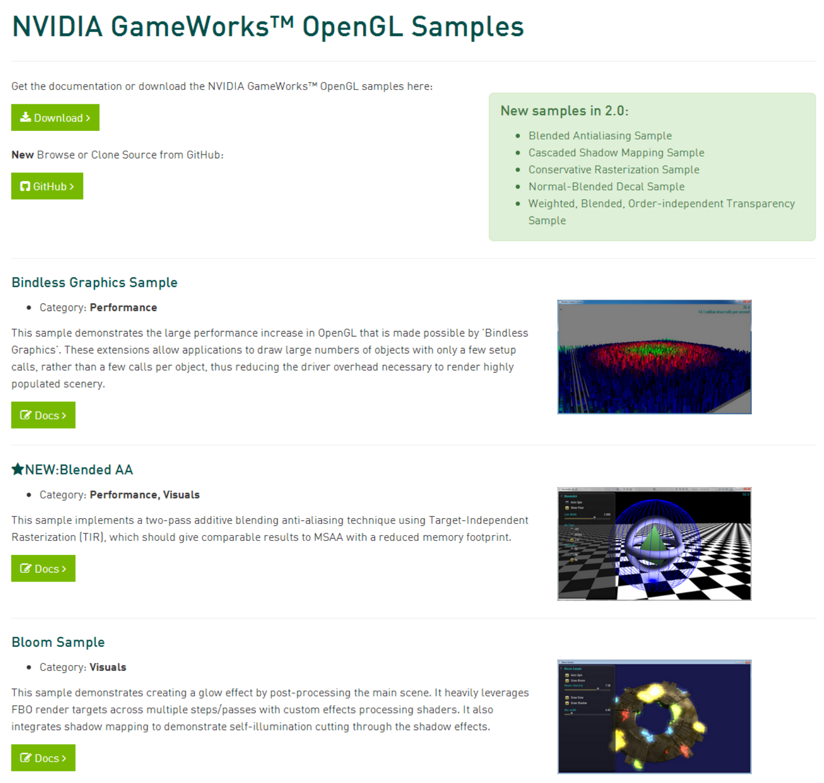
Source code: lightlab.c.

Snapshots: default (shown).

Samples

- **NVIDIA GameWorks™ Samples Overview**

  https://developer.nvidia.com/gameworks-samples-overview



NVIDIA GameWorks™ OpenGL Samples

Get the documentation or download the NVIDIA GameWorks™ OpenGL samples here:

Download ›

New Browse or Clone Source from GitHub:

GitHub ›

New samples in 2.0:

- Blended Antialiasing Sample
- Cascaded Shadow Mapping Sample
- Conservative Rasterization Sample
- Normal-Blended Decal Sample
- Weighted, Blended, Order-independent Transparency Sample

Bindless Graphics Sample

- Category: **Performance**

This sample demonstrates the large performance increase in OpenGL that is made possible by 'Bindless Graphics'. These extensions allow applications to draw large numbers of objects with only a few setup calls, rather than a few calls per object, thus reducing the driver overhead necessary to render highly populated scenery.

Docs ›

★NEW:Blended AA

- Category: **Performance, Visuals**

This sample implements a two-pass additive blending anti-aliasing technique using Target-Independent Rasterization (TIR), which should give comparable results to MSAA with a reduced memory footprint.

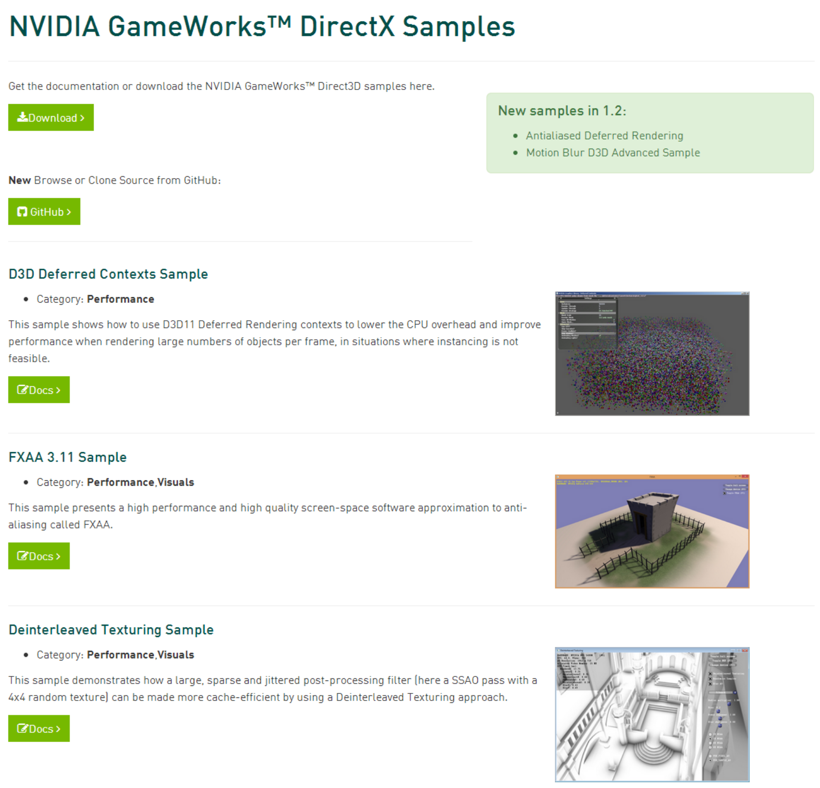
Docs ›

Bloom Sample

- Category: **Visuals**

This sample demonstrates creating a glow effect by post-processing the main scene. It heavily leverages FBO render targets across multiple steps/passes with custom effects processing shaders. It also integrates shadow mapping to demonstrate self-illumination cutting through the shadow effects.

Docs ›



NVIDIA GameWorks™ DirectX Samples

Get the documentation or download the NVIDIA GameWorks™ Direct3D samples here.

Download ›

New samples in 1.2:

- Antialiased Deferred Rendering
- Motion Blur D3D Advanced Sample

New Browse or Clone Source from GitHub:

GitHub ›

D3D Deferred Contexts Sample

- Category: **Performance**

This sample shows how to use D3D11 Deferred Rendering contexts to lower the CPU overhead and improve performance when rendering large numbers of objects per frame, in situations where instancing is not feasible.

Docs ›

FXAA 3.11 Sample

- Category: **Performance**,**Visuals**

This sample presents a high performance and high quality screen-space software approximation to anti-aliasing called FXAA.

Docs ›

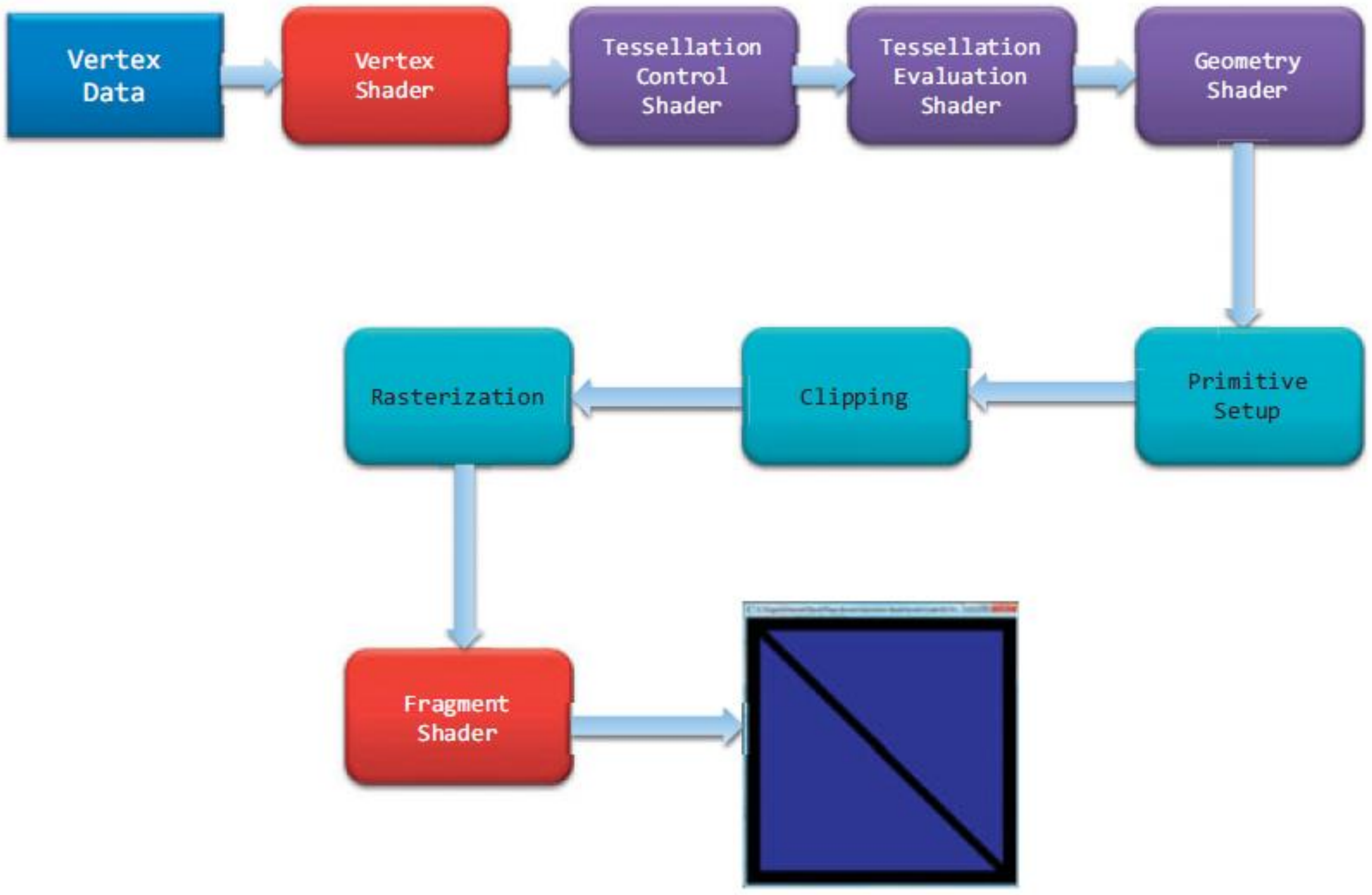
Deinterleaved Texturing Sample

- Category: **Performance**,**Visuals**

This sample demonstrates how a large, sparse and jittered post-processing filter (here a SSAO pass with a 4x4 random texture) can be made more cache-efficient by using a Deinterleaved Texturing approach.

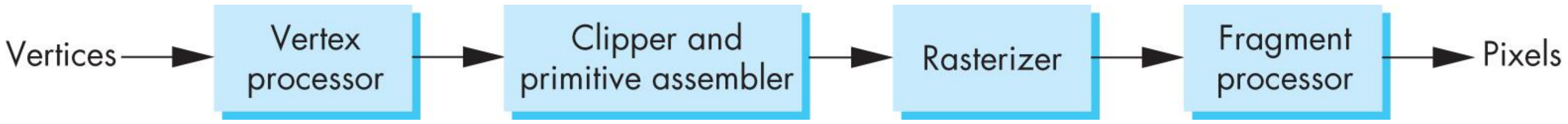
Docs ›

OpenGL Rendering Pipeline

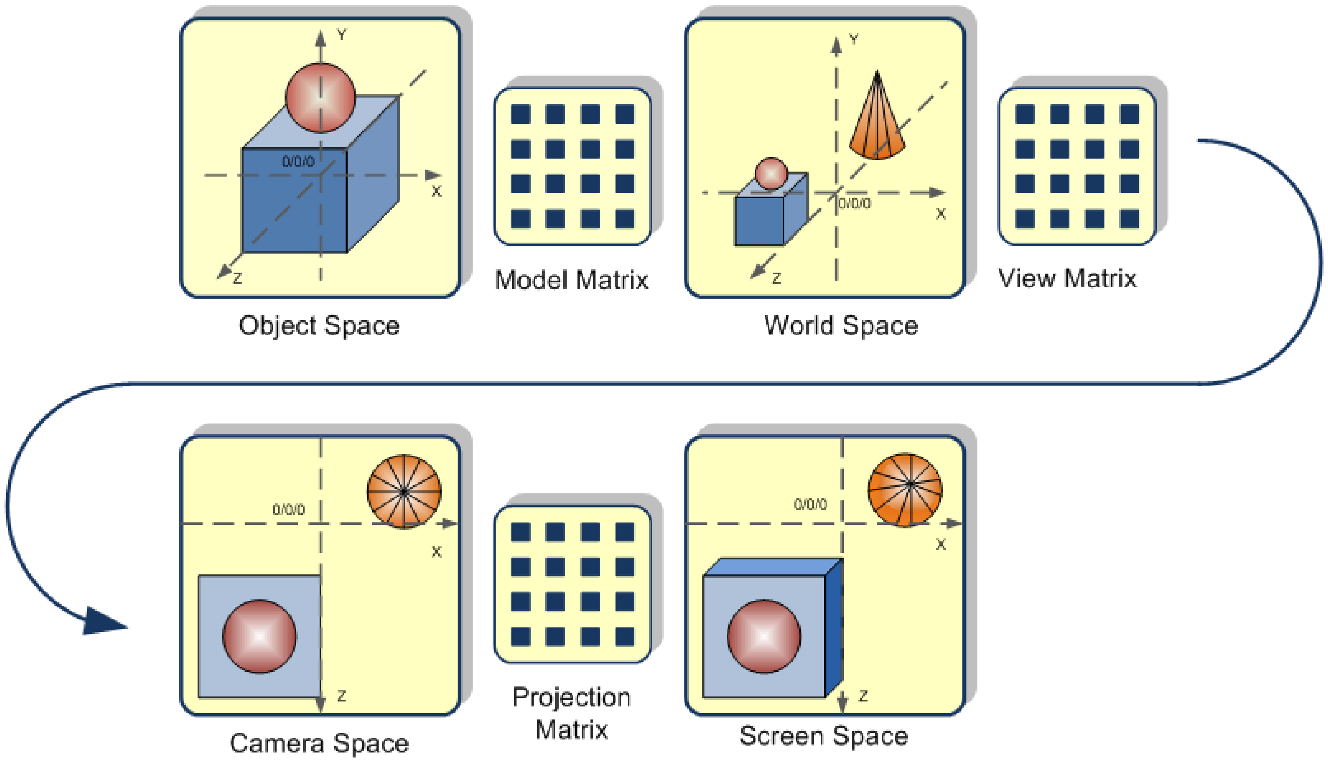


Vertex Processing

- Much of the work in the pipeline is in converting object representations from one coordinate system to another
    - Object coordinates
    - Camera (eye) coordinates
    - Screen coordinates
- Every change of coordinates is equivalent to a matrix transformation



Vertex Processing



1. Vertices of the Object to draw are in **Object space** (as modelled in your 3D Modeller)

2. … get transformed into World space by multiplying it with the **Model Matrix**

3. Vertices are now in **World space** (used to position the all the objects in your scene)

4. … get transformed into Camera space by multiplying it with the **View Matrix**

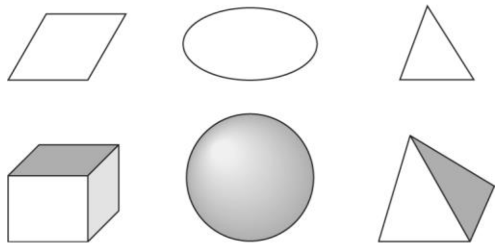
5. Vertices are now in **View Space** – think of it as if you were looking at the scene throught “the camera”

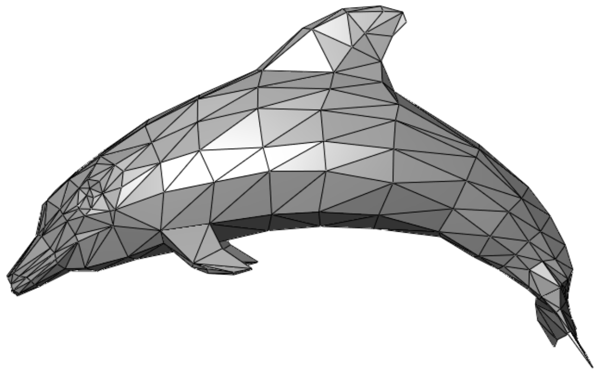
6. … get transformed into Screen space by multiplying it with the **Projection Matrix**

7. Vertex is now in **Screen Space** – This is actually what you see on your Display.

Geometric Primitives

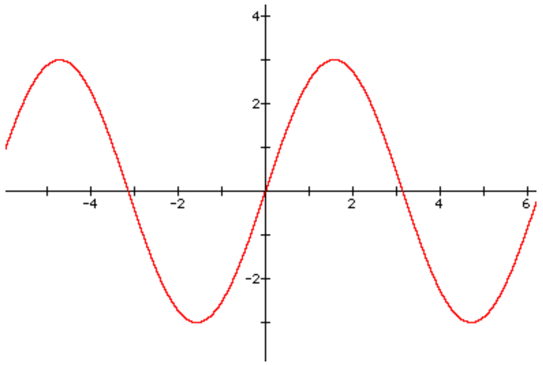
- Different philosophies:
  - Collections of complex shapes
    - Spheres, cones, cylinders, tori, …
  - One simple type of geometric primitive
    - Triangles or triangle meshes
  - Small set of complex primitives with adjustable parameters
    - E.g. "all polynomials of degree 2"
    - Splines, NURBS (details in CPSC 424)
    - Implicits



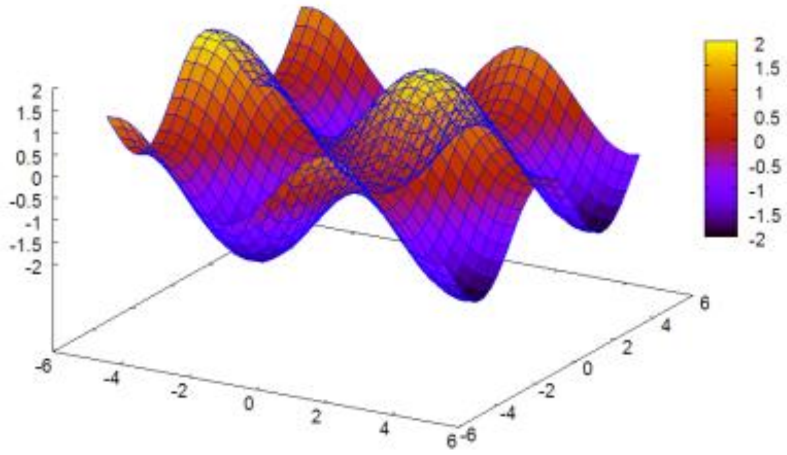


Geometric Primitives

- Explicit Functions
- Curves:
  - y is a function of x: $y:=\sin(x)$
  - Only works in 2D



- Surfaces:
  - z is a function of x and y: $z:=\sin(x)+\cos(y)$
  - Cannot define arbitrary shapes in 3D



Geometric Primitives

- Parametric Functions
- Curves:
  - 2D: x and y are functions of a parameter value t
  - 3D: x, y, and z are functions of a parameter value t

$$C(t) := \begin{pmatrix} \cos(t) \\ \sin(t) \\ t \end{pmatrix}$$

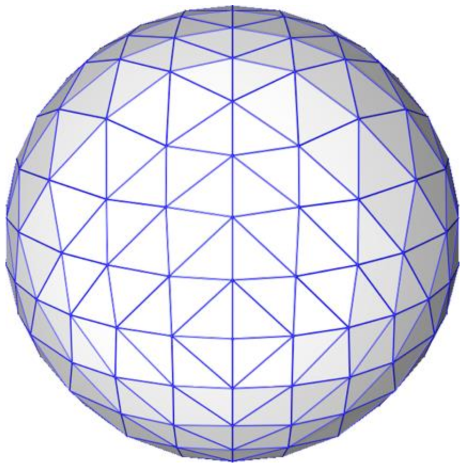
- Surfaces:
  - Surface S is defined as a function of parameter values s, t
  - Names of parameters can be different to match intuition

$$S(\phi, \theta) := \begin{pmatrix} \cos(\phi)\cos(\theta) \\ \sin(\phi)\cos(\theta) \\ \sin(\theta) \end{pmatrix}$$

Geometric Primitives

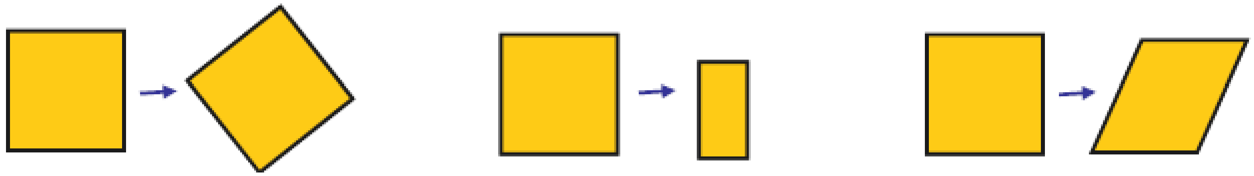
- Implicit Functions
- Surface:
  - Surface defined by zero set (roots) of function
  - E.g.

$$S(x,y,z): x^2 + y^2 + z^2 - 1 = 0$$

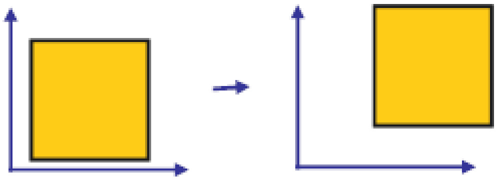


Model/View Transformation

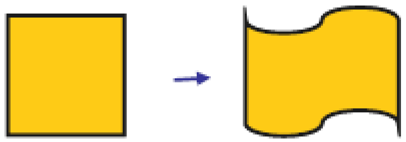
- Types of transformations:
  - Rotations, scaling, shearing



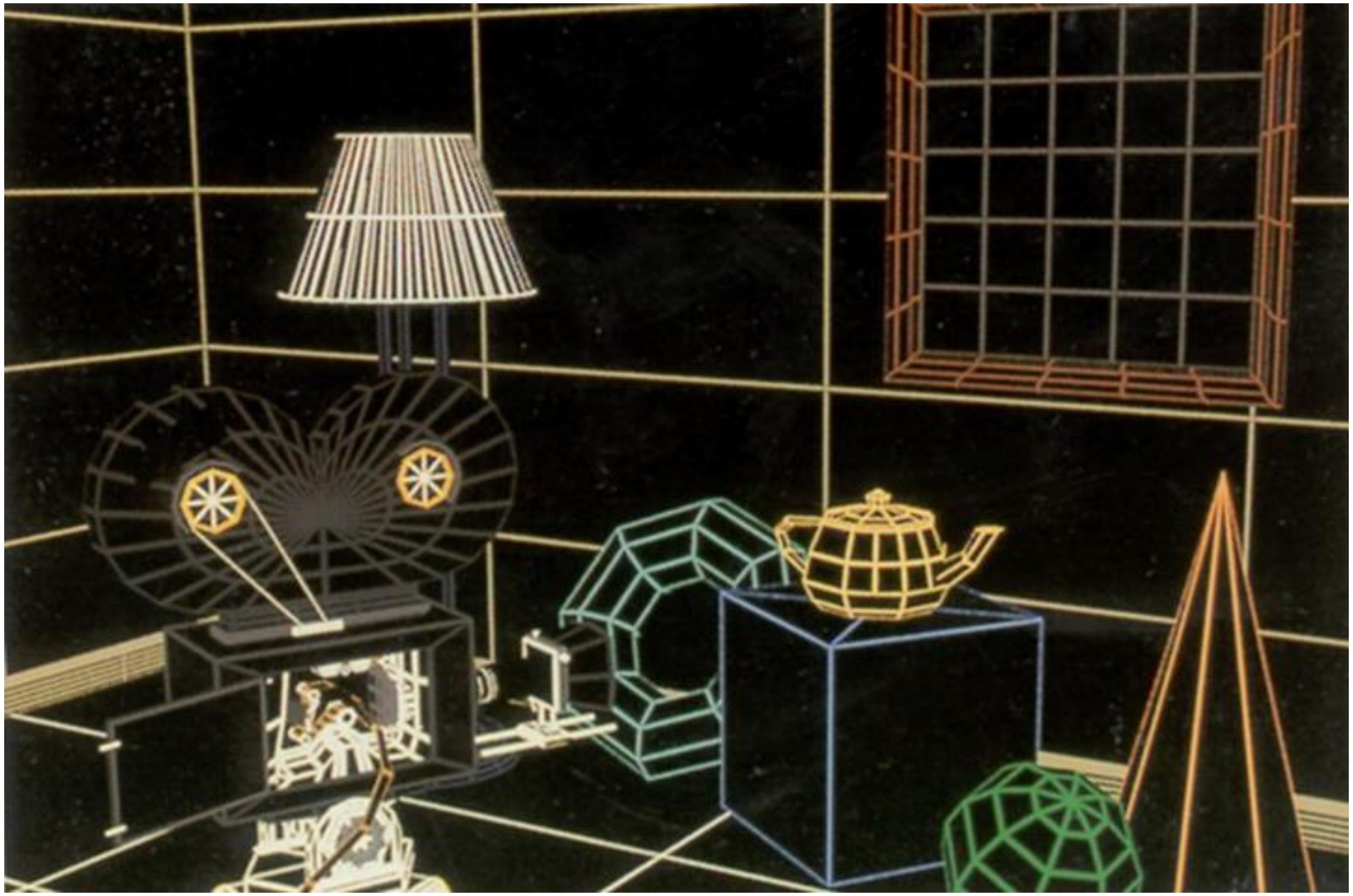
  - Translations



  - Other transformations (not handled by rendering pipeline)
    - Freeform deformation

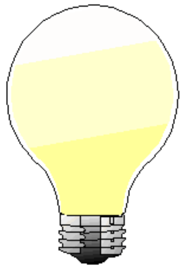


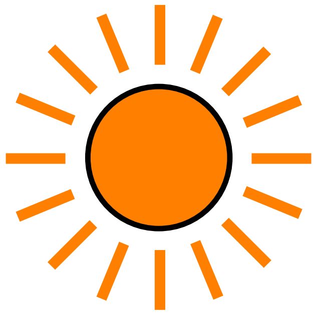
Example - Modeling and Viewing Transformations

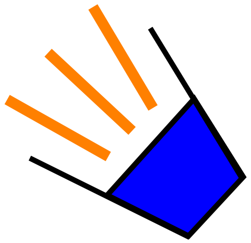


Lighting

- Compute brightness based on property of material and light position(s)
- Computation is performed per-vertex
- There are several kinds of lights (or light sources)
  - Light bulbs
  - The sun
  - Spot Lights
  - Ceiling Lights



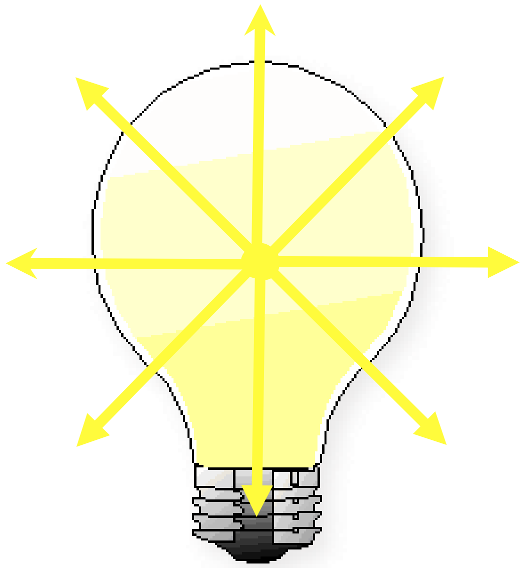




- These are different because they emit photons (光子) differently

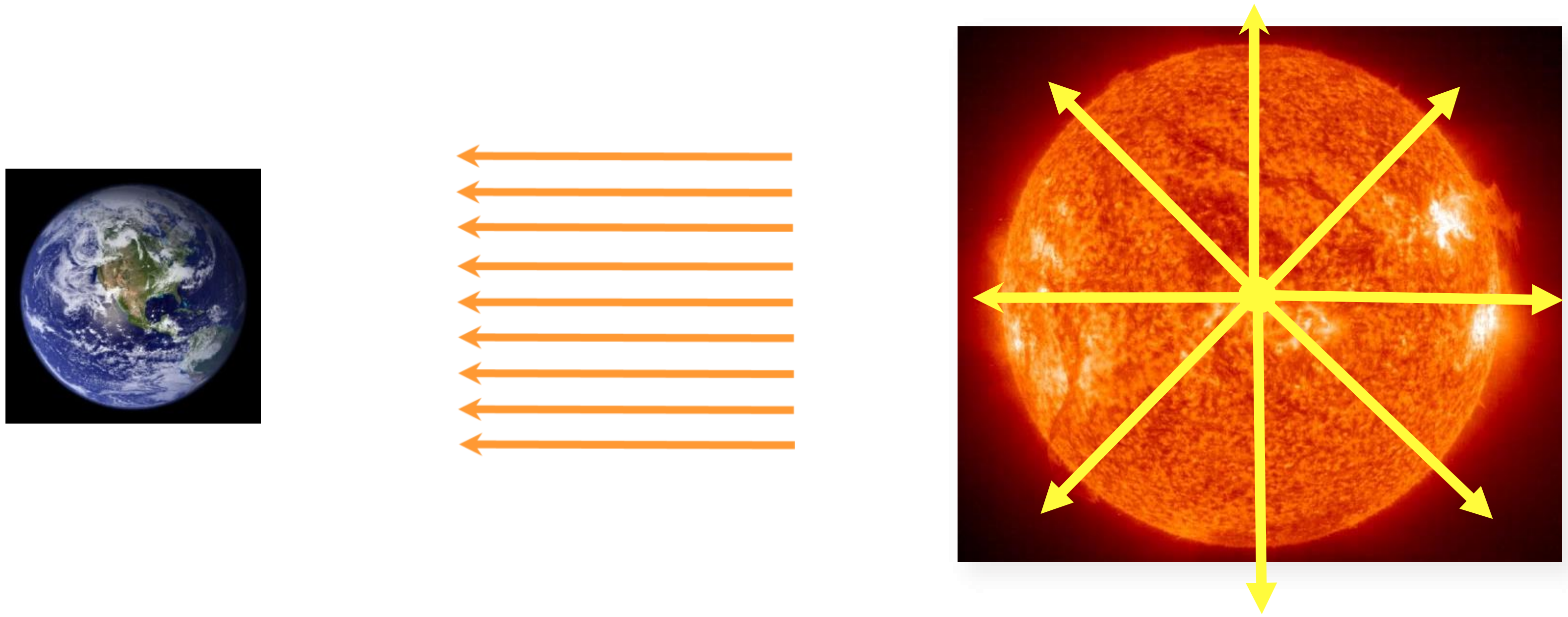
Point Lights (i.e. Light Bulbs)

- Emit light evenly in all directions.
- That is, photons (rays) from a point light all originate from the same point, and have directions evenly distributed over the sphere.



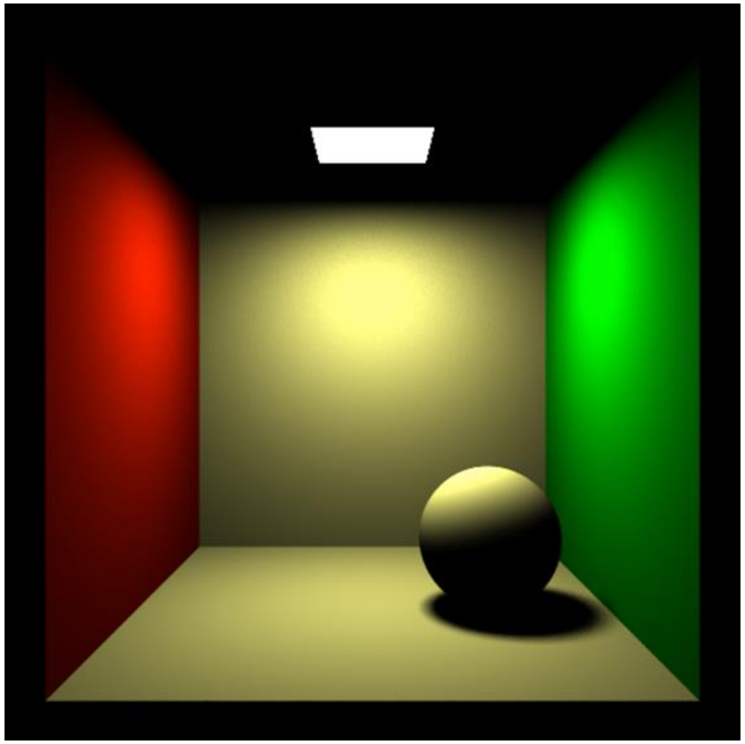
Directional Lights (i.e. the Sun)

- Point light sources at an infinite (or near infinite) distance.
- Can assume that all rays are parallel.

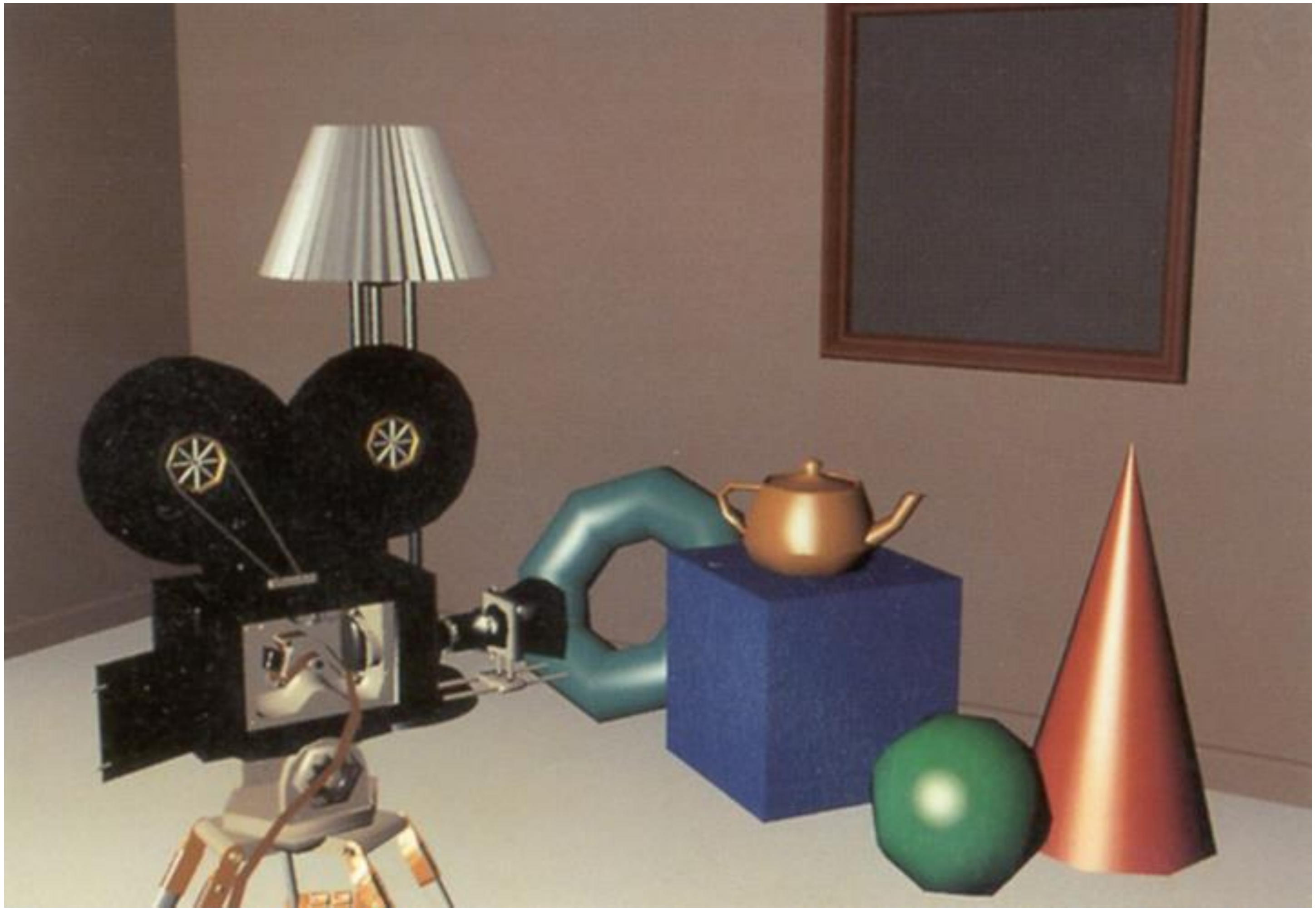


Area Lights (i.e. Ceiling Lights)

- Emits light in every direction from a surface
- Can think of it as a set of point lights, or a patch on which every point is a point light



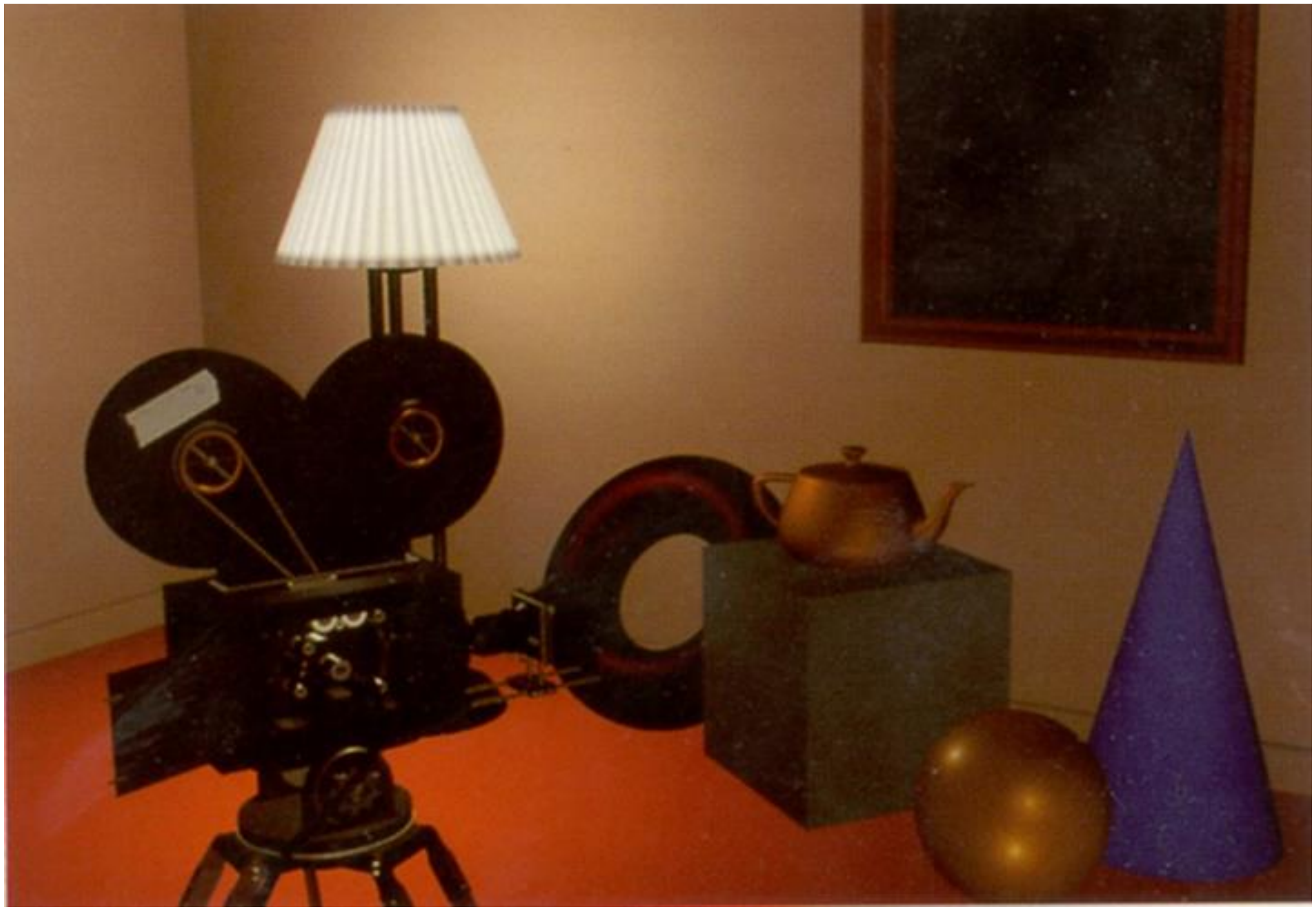
Example - Lighting



Shading

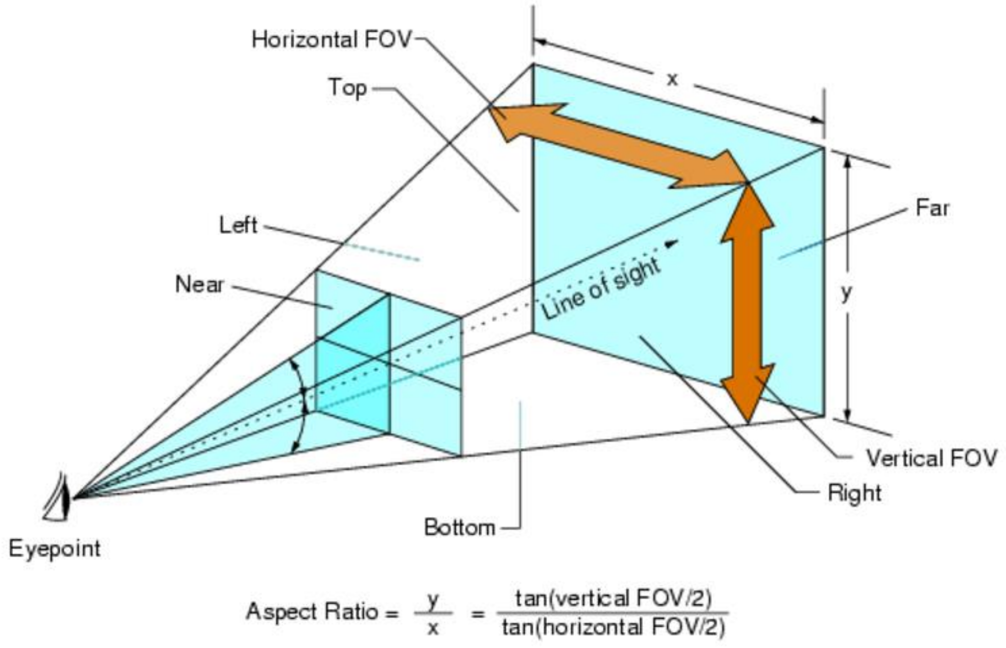
- Problem: How do we determine the color of a piece of geometry?
- In the real world, color depends on the object’s surface color and the color of the light.
- It is the same way in computer graphics.
- “Shading” is the process by which color is assigned to geometry.

Example - Complex Lighting and Shading



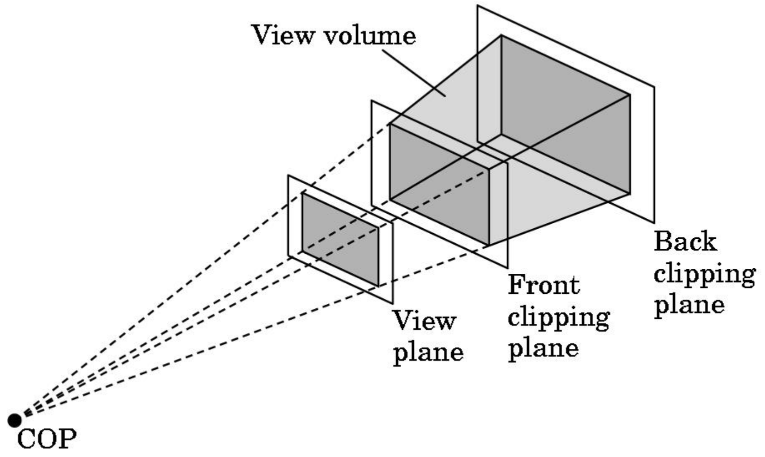
3D to 2D (Projection)

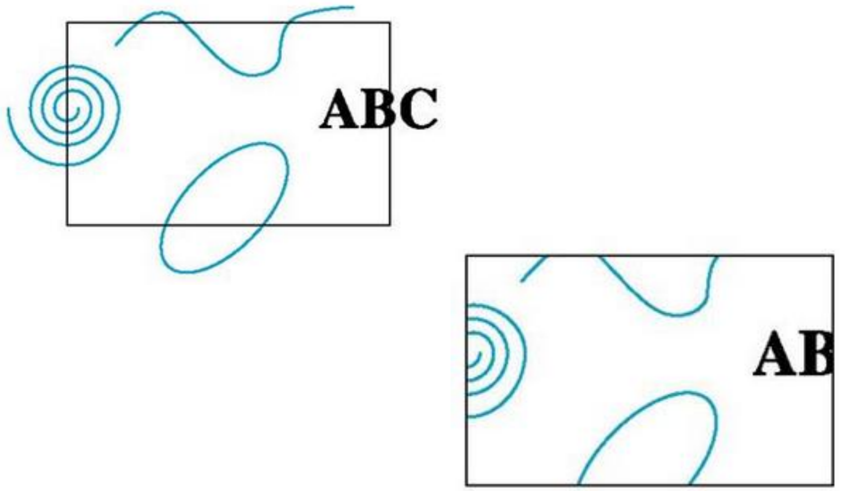
- Problem: The display is, virtually always, only 2D
  - Need to transform the 3D model into 2D
- We do this with a virtual camera
- Represented mathematically by a 3x4 projection (or P) matrix



Clipping

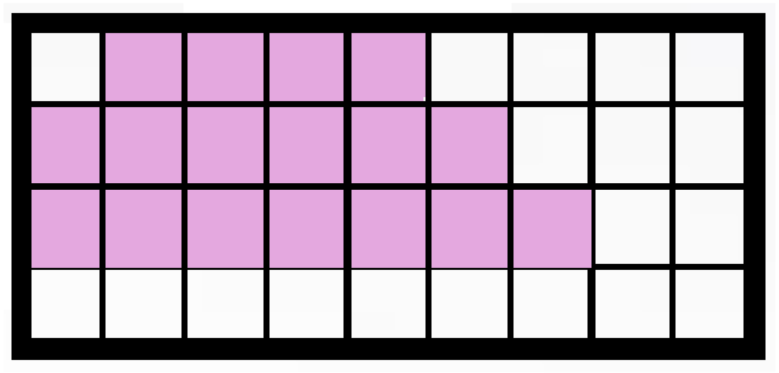
- Problem: The camera doesn't see the whole scene
    - In particular, the camera might only see parts of objects
- Solution: Find objects that cross the edge of the viewing volume, and "clip" them
    - Clip: Cut a polygon into multiple parts, such that each is entirely inside or outside the display area





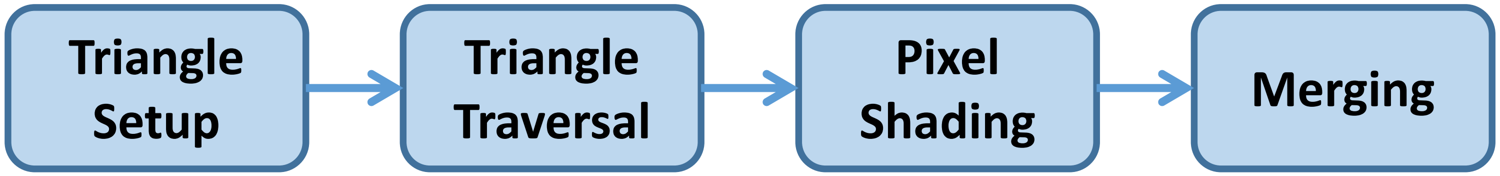
Scan conversion

- Convert continuous 2D geometry (lines, polygons etc.) to discrete
- Raster display – discrete grid of elements



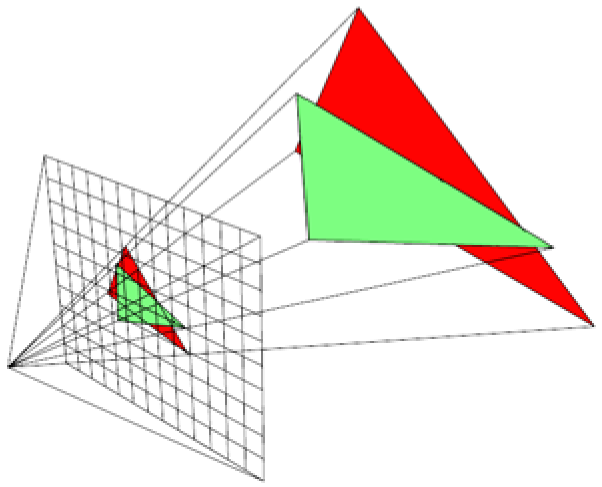
Rasterizer Stage

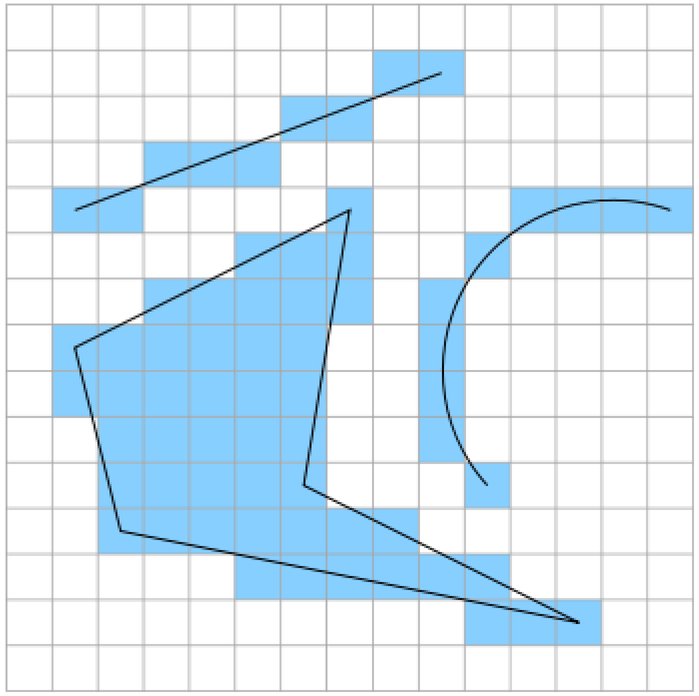
- Rasterizer
  - Conversion from two-dimensional vertices in screen space – each with a z-value (depth value), and various shading information associated with each vertex – into pixels on the screen (On GPU)



Related Terminology

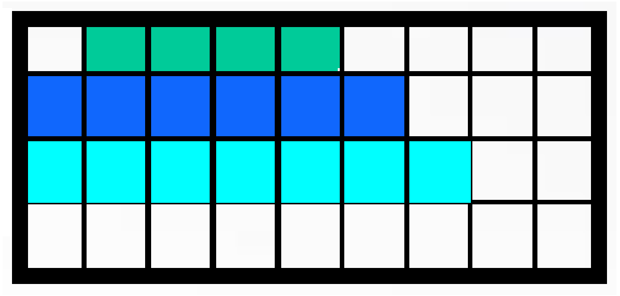
- Resolution: number of rows & columns in device
- Screen Space: Discrete 2D Cartesian coordinate system of the screen pixels



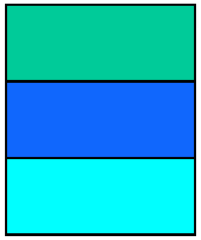


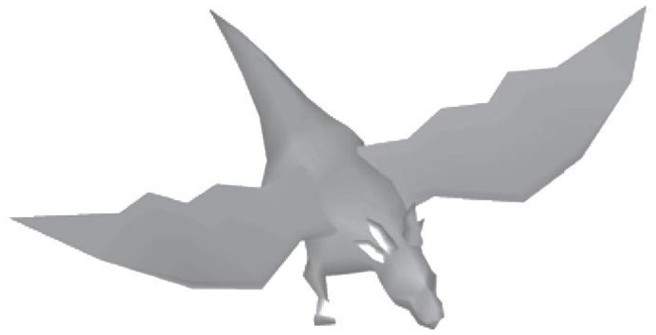
Texture mapping

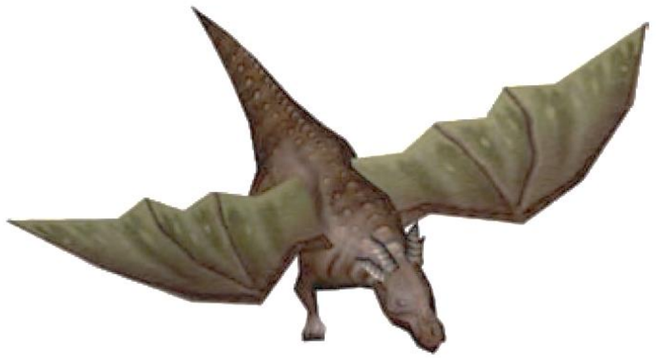
- “Gluing (粘合) images onto geometry”
- Color of every fragment is altered by looking up a new color value from an image.

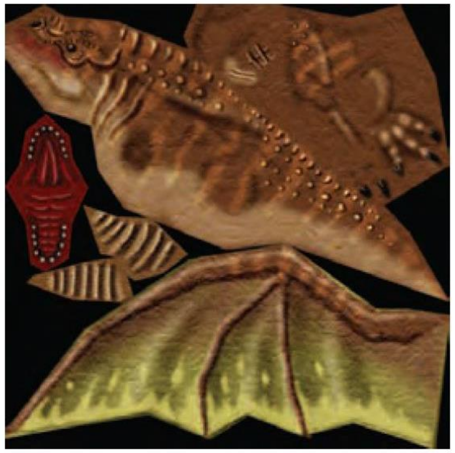


Image



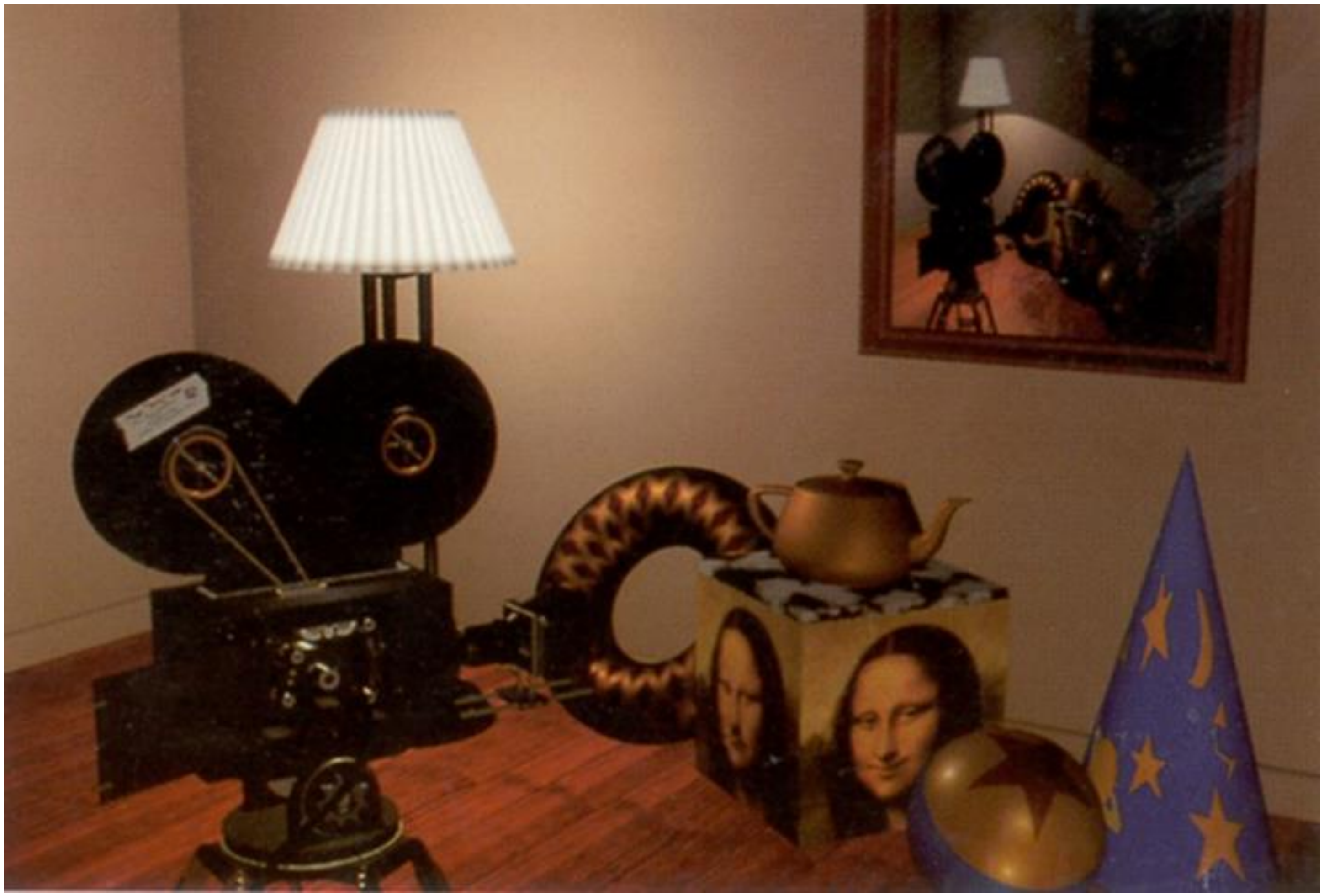






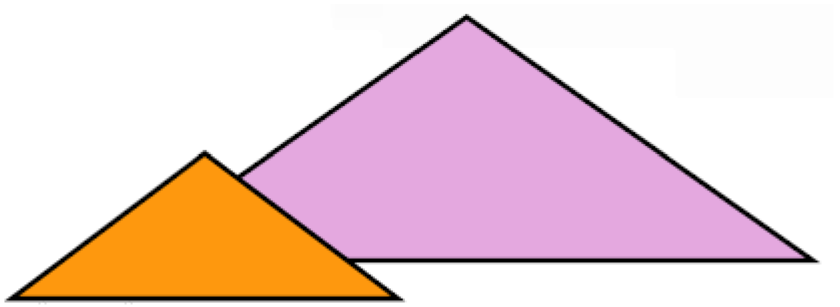
Skin Image

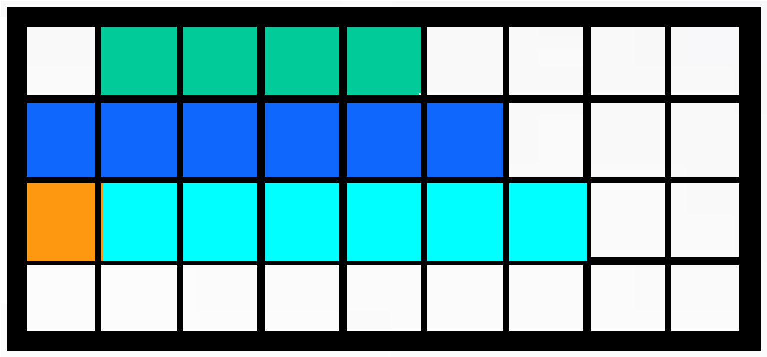
Example – Texture Mapping



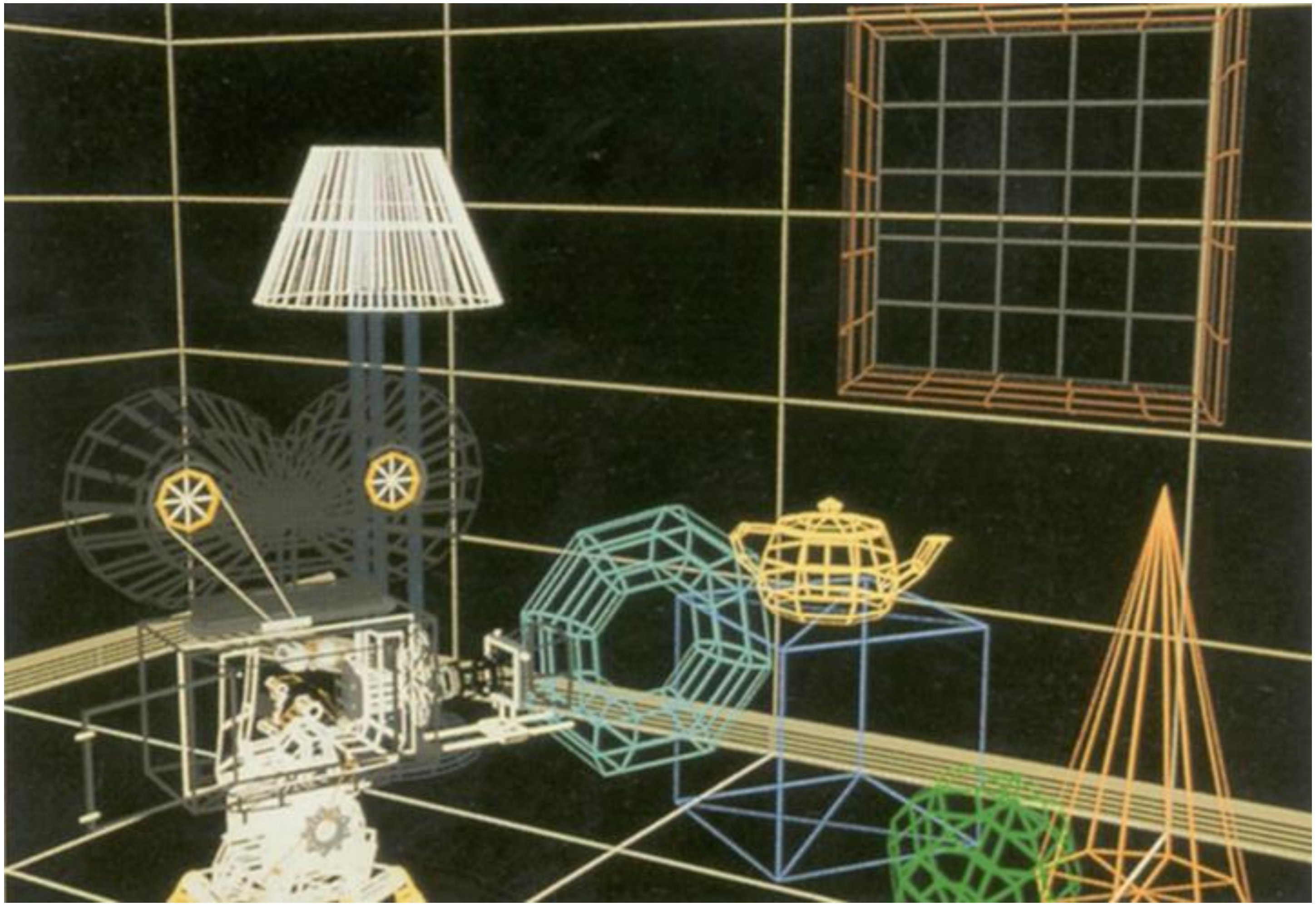
Depth Test

- Remove parts of geometry hidden behind other geometric objects
- Perform on every individual fragment

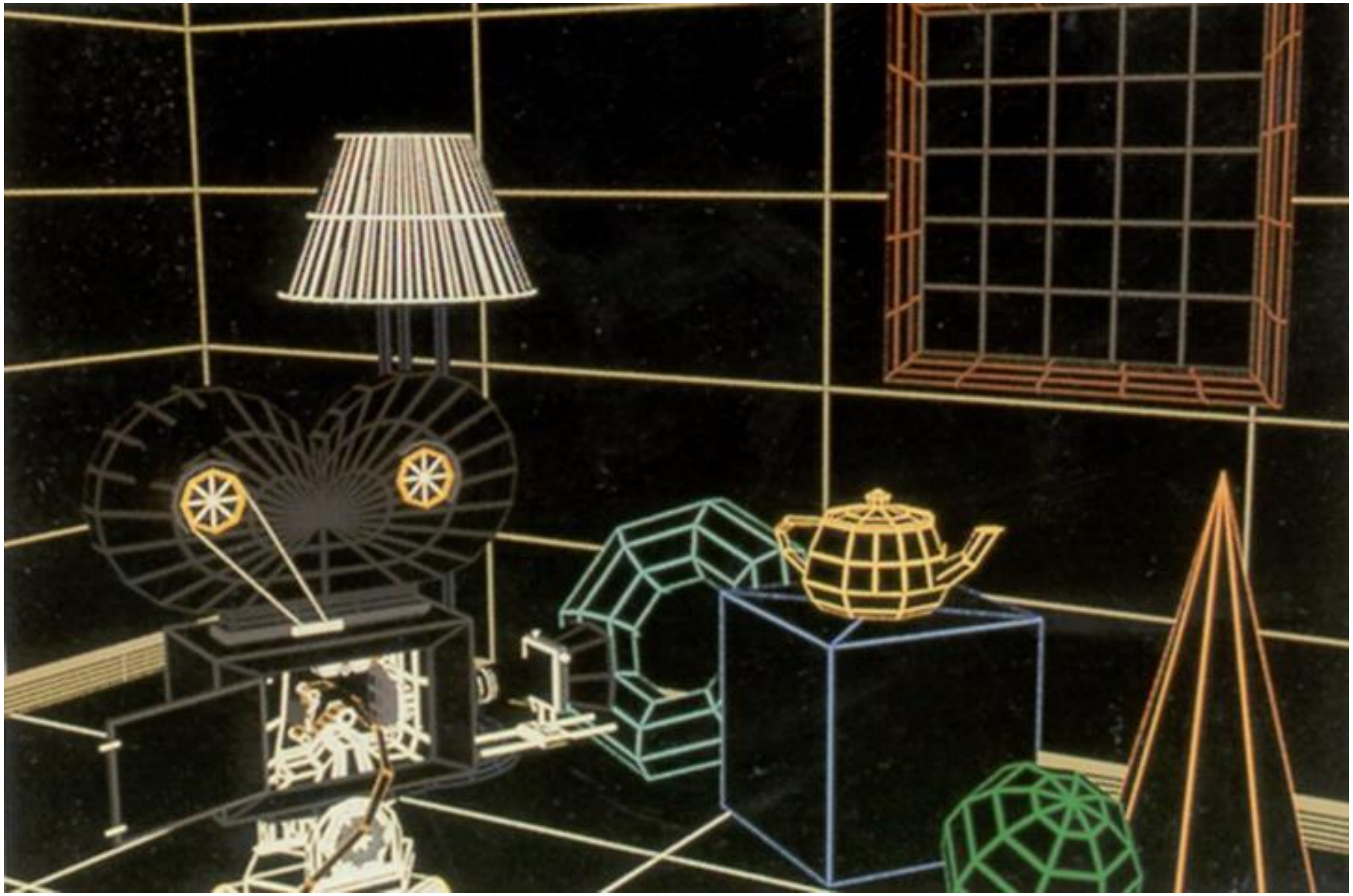




Example - Without Hidden Line Removal

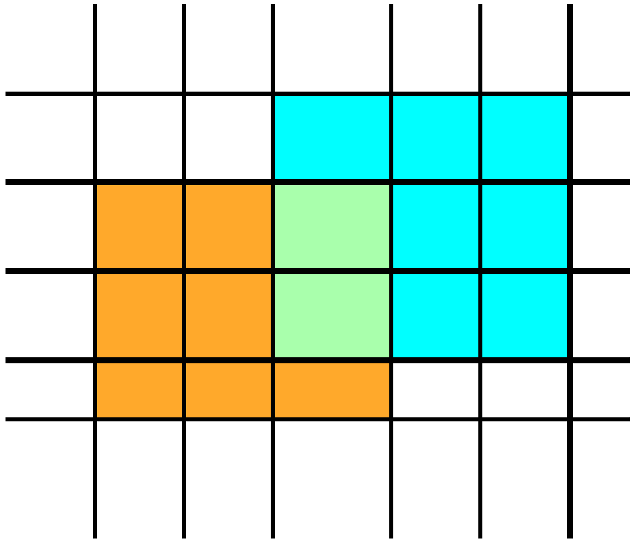


Example - Hidden Line Removal



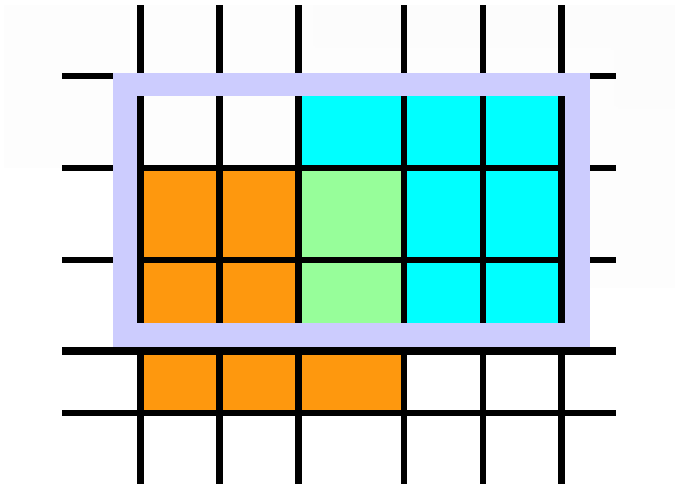
Blending

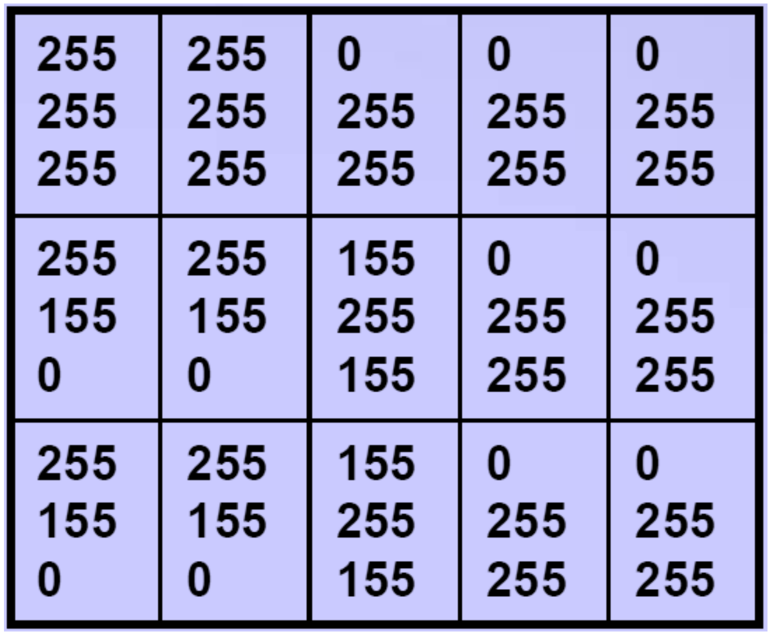
- final image: write fragments to pixels
- draw from farthest to nearest
  - no blending – replace previous color
  - blending: combine new & old values with arithmetic operations



Frame-buffer

- video memory(图象存储器) on graphics board that holds image
- double-buffering: two separate buffers
  - draw into one while displaying other, then swap to avoid flicker





255 255 255	255 255 255	0 255 255	0 255 255	0 255 255
255 155 0	255 155 0	155 255 155	0 255 255	0 255 255
255 155 0	255 155 0	155 255 155	0 255 255	0 255 255

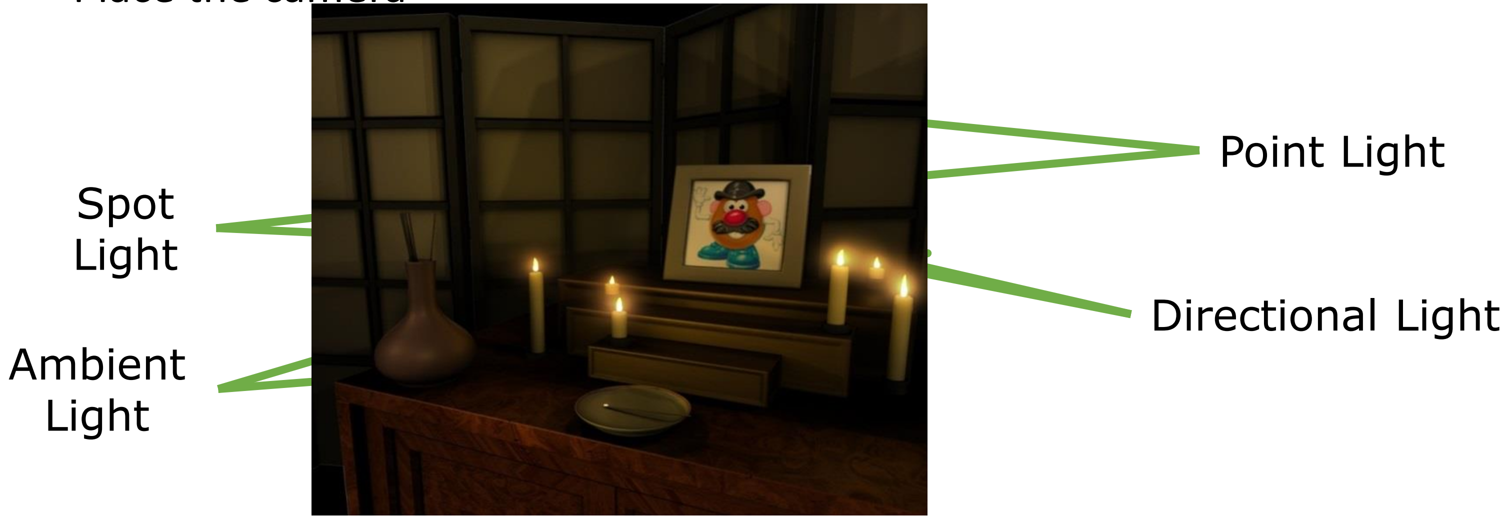
Modeling vs. Rendering

- **Modeling**
  - Create models
  - Apply materials to models
  - Place models around scene
  - Place lights in scene
  - Place the camera

- **Rendering**

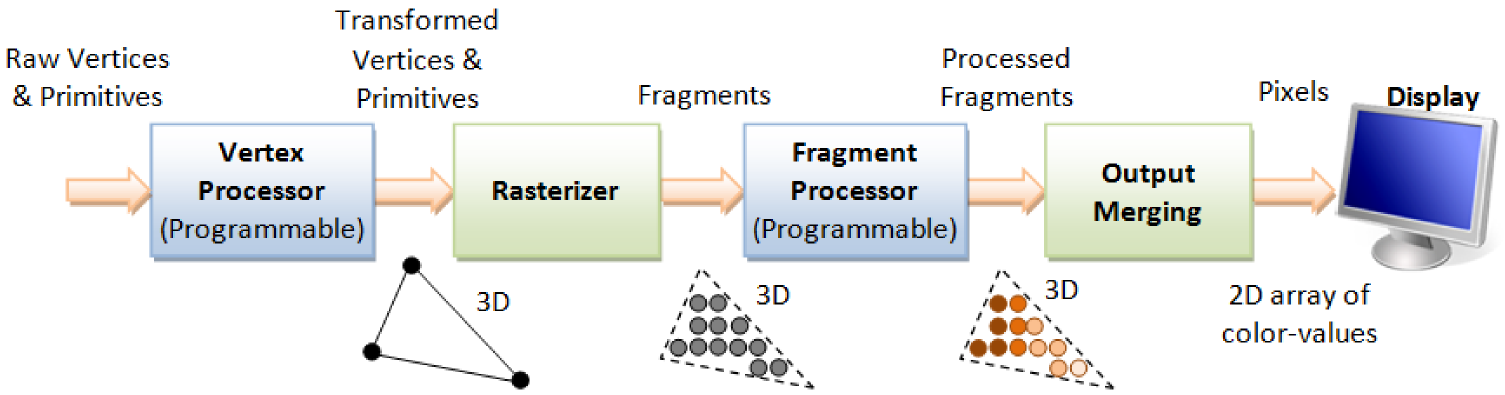
  Take "picture" with camera
- Both can be done with commercial software:

  Autodesk Maya^{TM} ,3D Studio Max^{TM}, Blender^{TM}, etc



Summary

- The 3D graphics rendering pipeline consists of the following main stages:
    - **Vertex Processing**: Process and transform individual vertices.
    - **Rasterization**: Convert each primitive (connected vertices) into a set of fragments. A fragment can be treated as a pixel in 3D spaces, which is aligned with the pixel grid, with attributes such as position, color, normal and texture.
    - **Fragment Processing**: Process individual fragments.
    - **Output Merging**: Combine the fragments of all primitives (in 3D space) into 2D color-pixel for the display.


Raw Vertices & Primitives
Vertex Processor (Programmable)
Transformed Vertices & Primitives
Rasterizer
Fragments
Fragment Processor (Programmable)
Processed Fragments
Output Merging
Pixels
Display
3D
3D
3D
2D array of color-values

3D Graphics Rendering Pipeline: Output of one stage is fed as input of the next stage. A vertex has attributes such as (x, y, z) position, color (RGB or RGBA), vertex-normal (n_x, n_y, n_z), and texture. A primitive is made up of one or more vertices. The rasterizer raster-scans each primitive to produce a set of grid-aligned fragments, by interpolating the vertices.