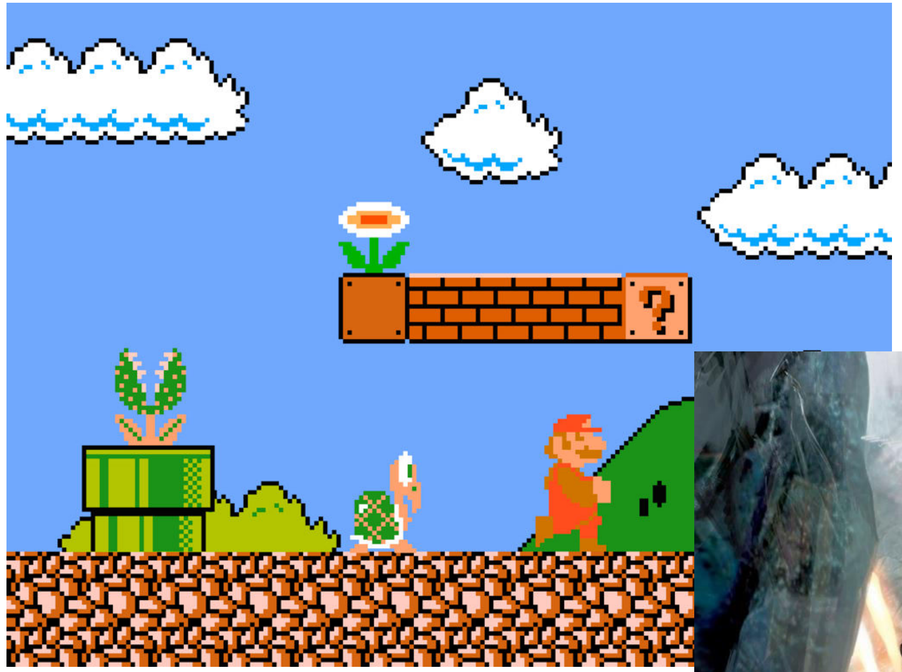
Introduction

CSCI 3800/6820: Computer Graphics

3D Graphics Everywhere!

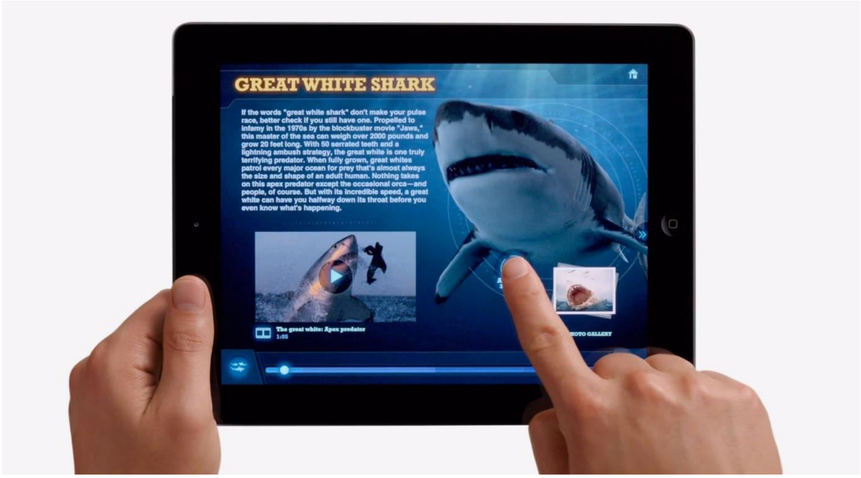
Games



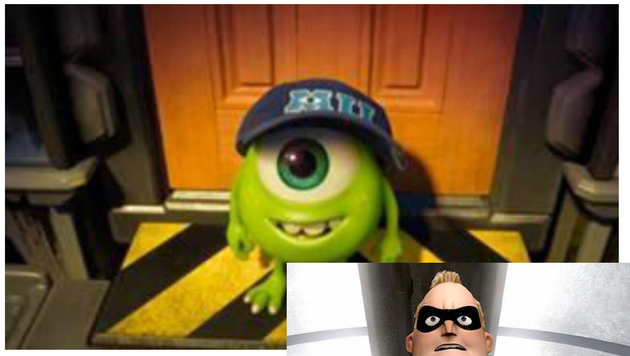
Halo 3 developer: Bungie Software publisher: Microsoft
© GameWallpapers.com hosted by JTLnet.com

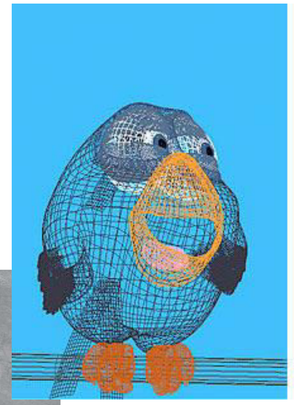
XP
HAPPY HALLOWEEN
GOTHAM CITY
Donuts
SIGNAL LOCK
Line Launcher Perch (Hold)

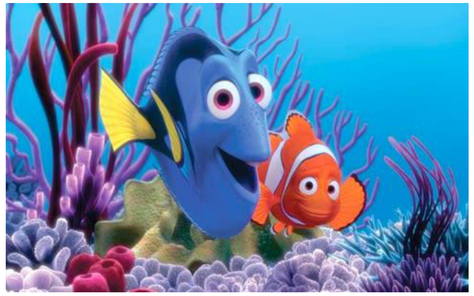
Graphical Interaction

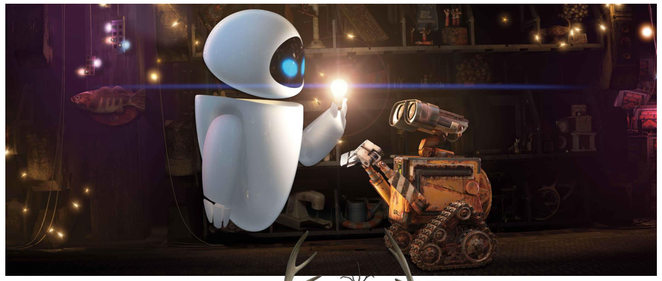


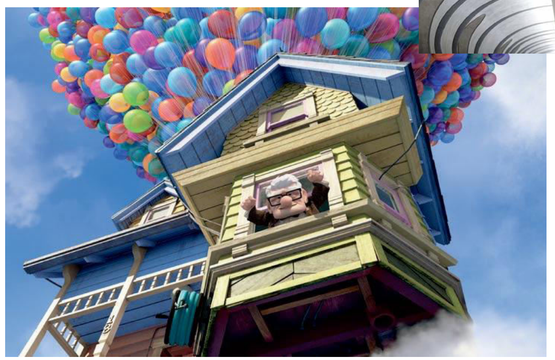
Animated Films







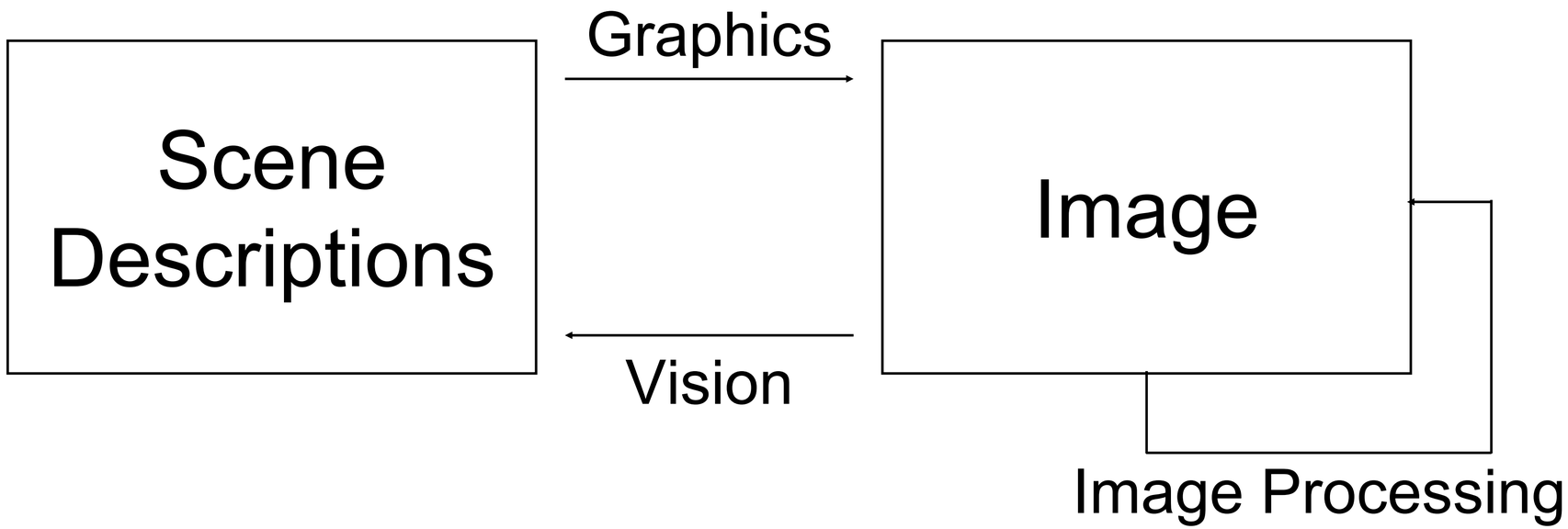




Special Effects

Virtual/Augmented Reality

Graphics, Image Processing, and Vision



Math

Geometry

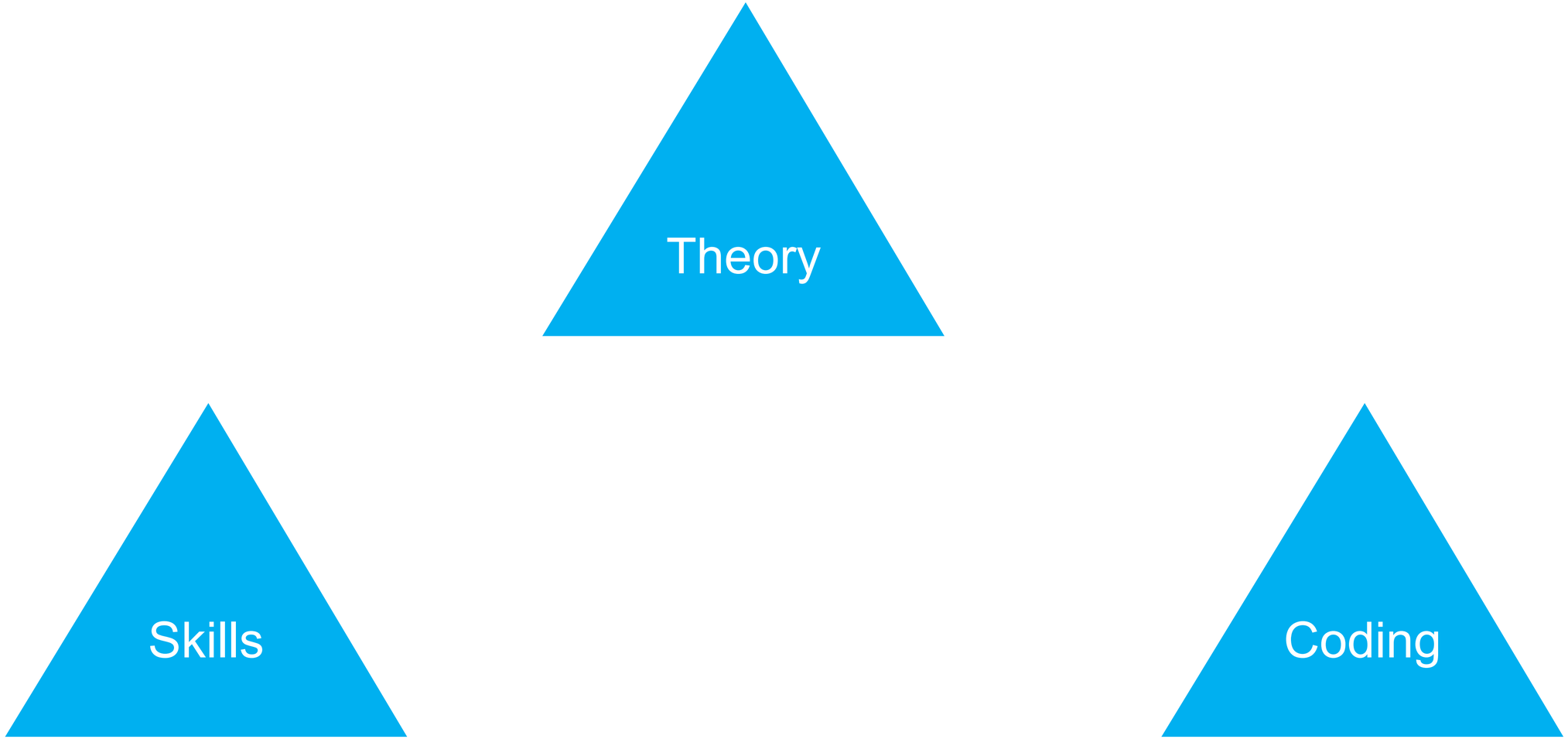
~~3D Graphics~~ Everywhere!

Algorithms

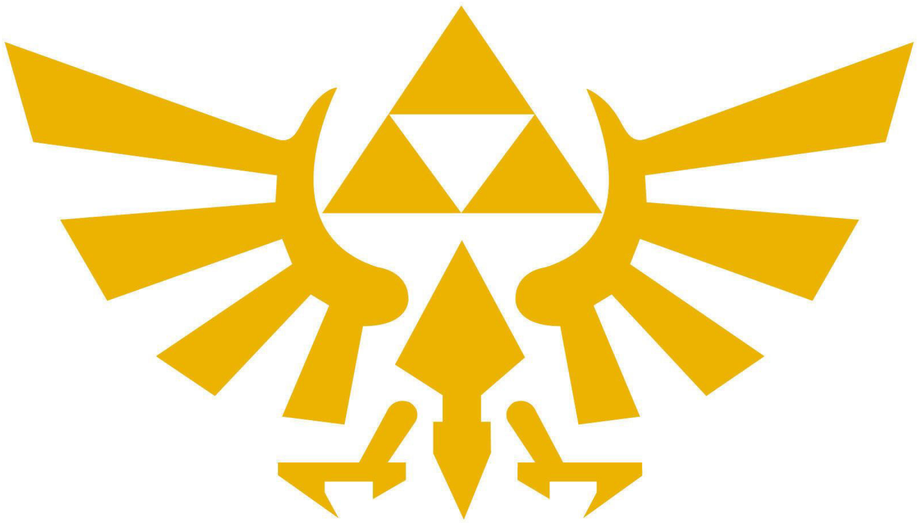
What will this class cover?

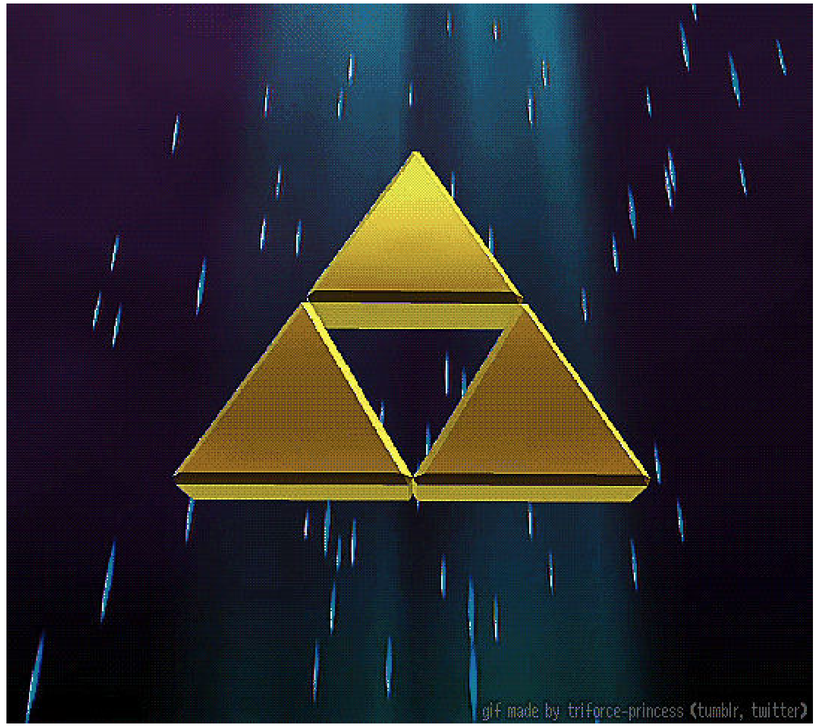
- **Skills**
  - 3D Modeling
  - Texture Painting
  - Lighting
  - Creating Environments
- **Theory**
  - Perspective Projection
  - The Rendering Pipeline
  - The Lighting Equation
  - Ray Tracing
- **Coding**
  - OpenGL
  - Vertex Shaders
  - Fragment Shaders
  - Radiance Fields

It is powerful stuff!



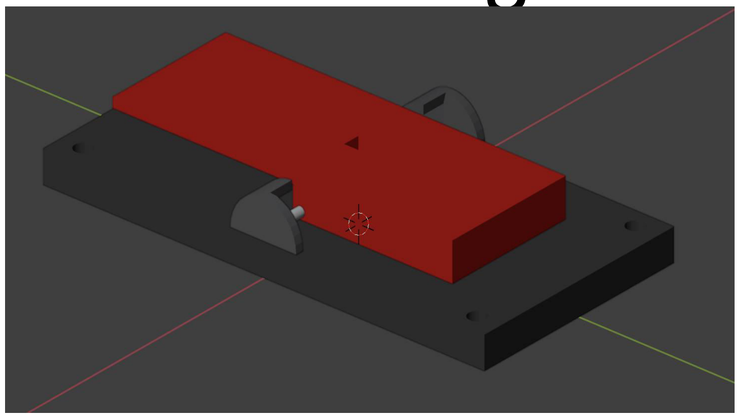
Think of it like the Triforce…





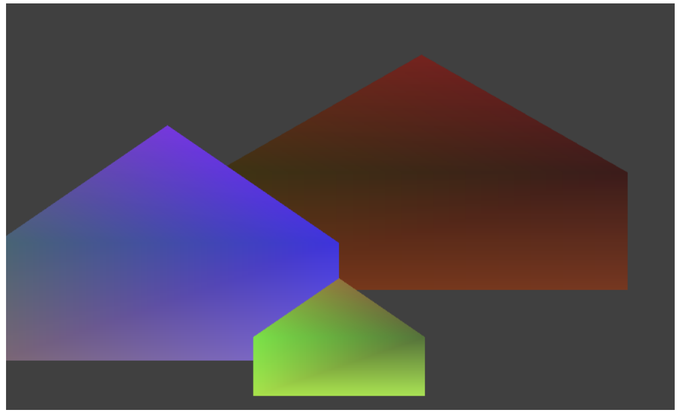
What You're Going to Build

3D Modeling Lab



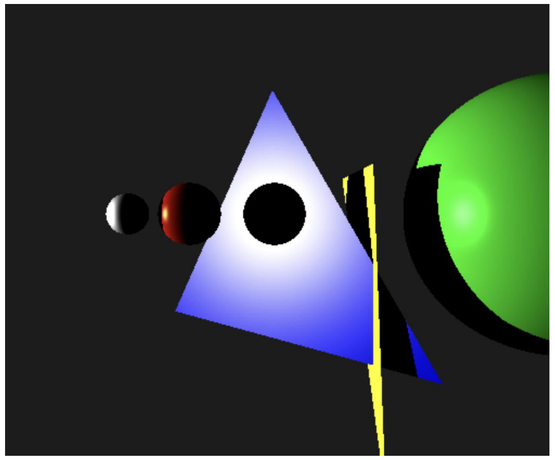
Projection Lab

Shaders Lab



3D Environment Lab

Ray Tracing Lab

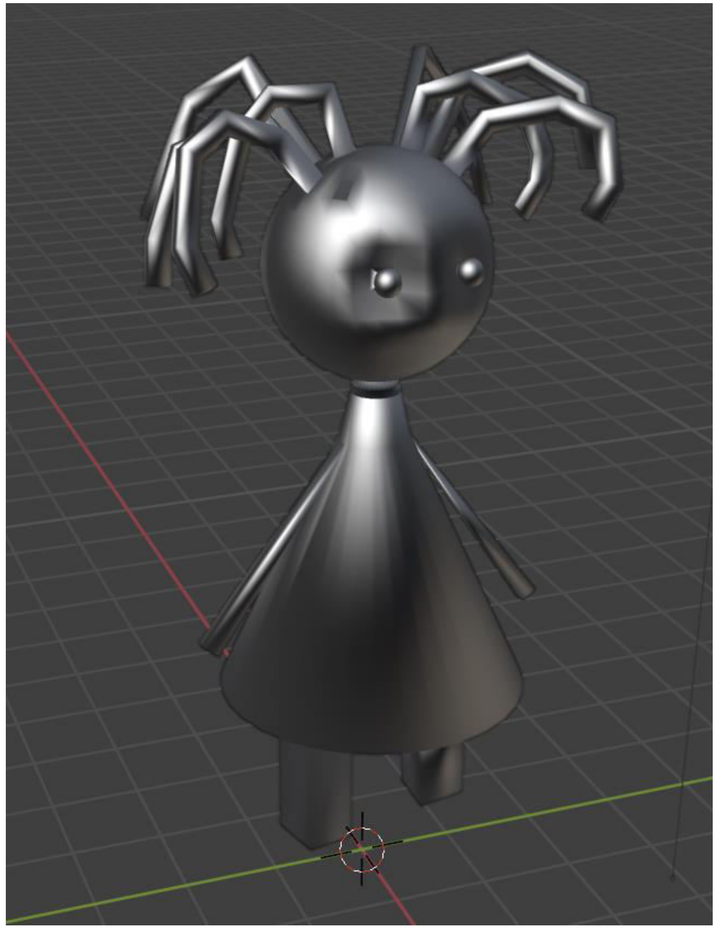


Final Project

You will complete a final project of your choosing that extends the ideas of this course. Possible final projects include:

- Exploring advanced concepts in 3D modeling
- Making a game with 3D content using Unity/Unreal
- Increasing the capabilities of your ray tracer
- Building an application that utilizes OpenGl

Note: This course will not turn you into a 3D artist over night…



Dr. Hart’s First 3D Model

My Research

Fluid Simulation

https://www.youtube.com/watch?v=CWrTJ7ikQzw

Style Transfer

https://davidmhart.github.io/interpolated-selectionconv/

Syllabus

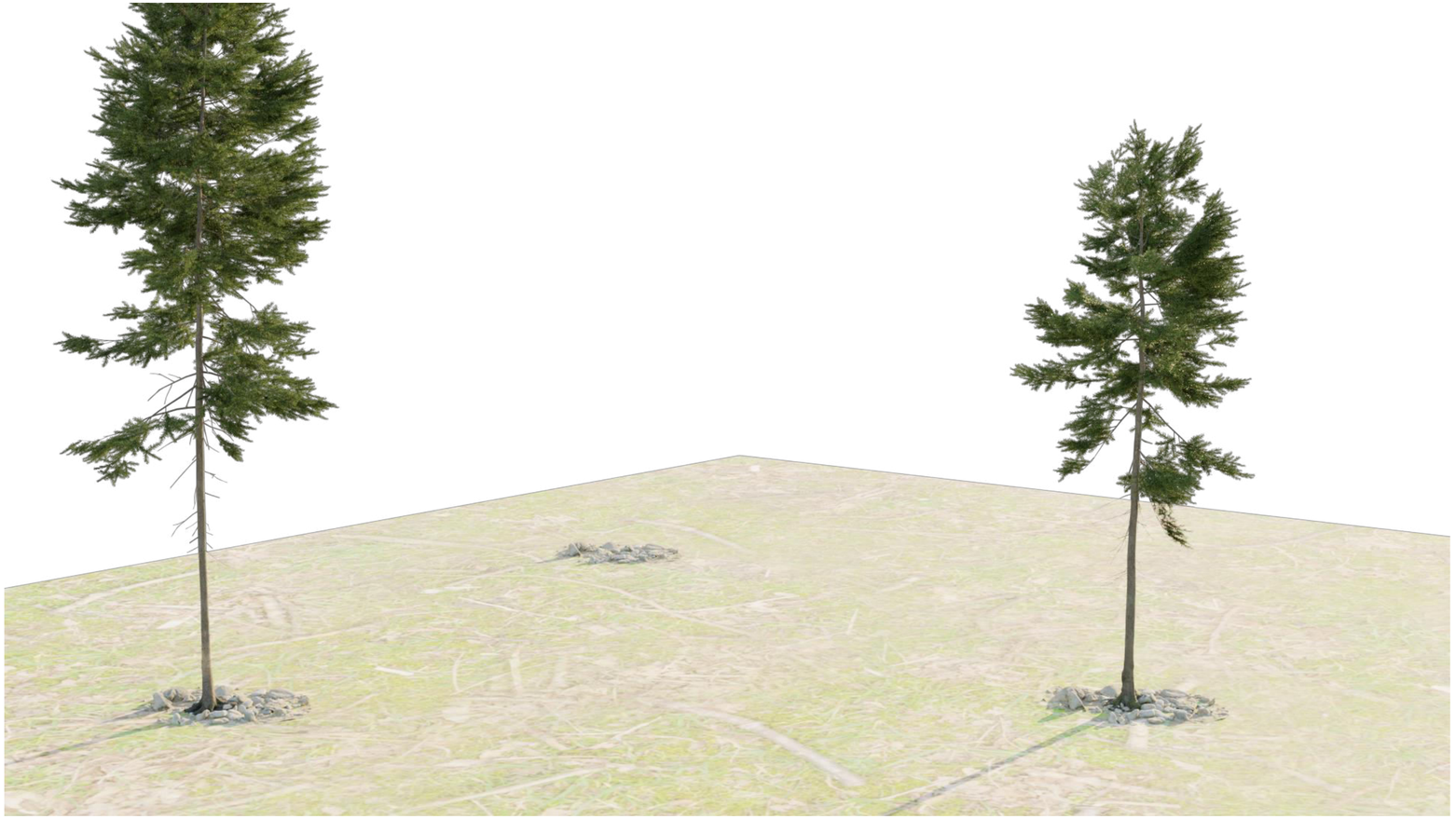
- Prerequisites
- Policies
- Schedule
- Assignments
- Practice Quizzes
- Final Project

Software for this Course

- Blender for Labs 1 and 2
  - Free Download
  - On Austin 208 Machines
  - Use a 3-Button Mouse!
- Python with Conda for Lab 3
- Python or C++ for Lab 4
- TBD for Lab 5

Demo: Blender Basics

- Navigating Blender
- Viewport
- Camera
- Rendering



Representing 3D Objects and Scenes

CSCI 3800/6820: Computer Graphics

Outline

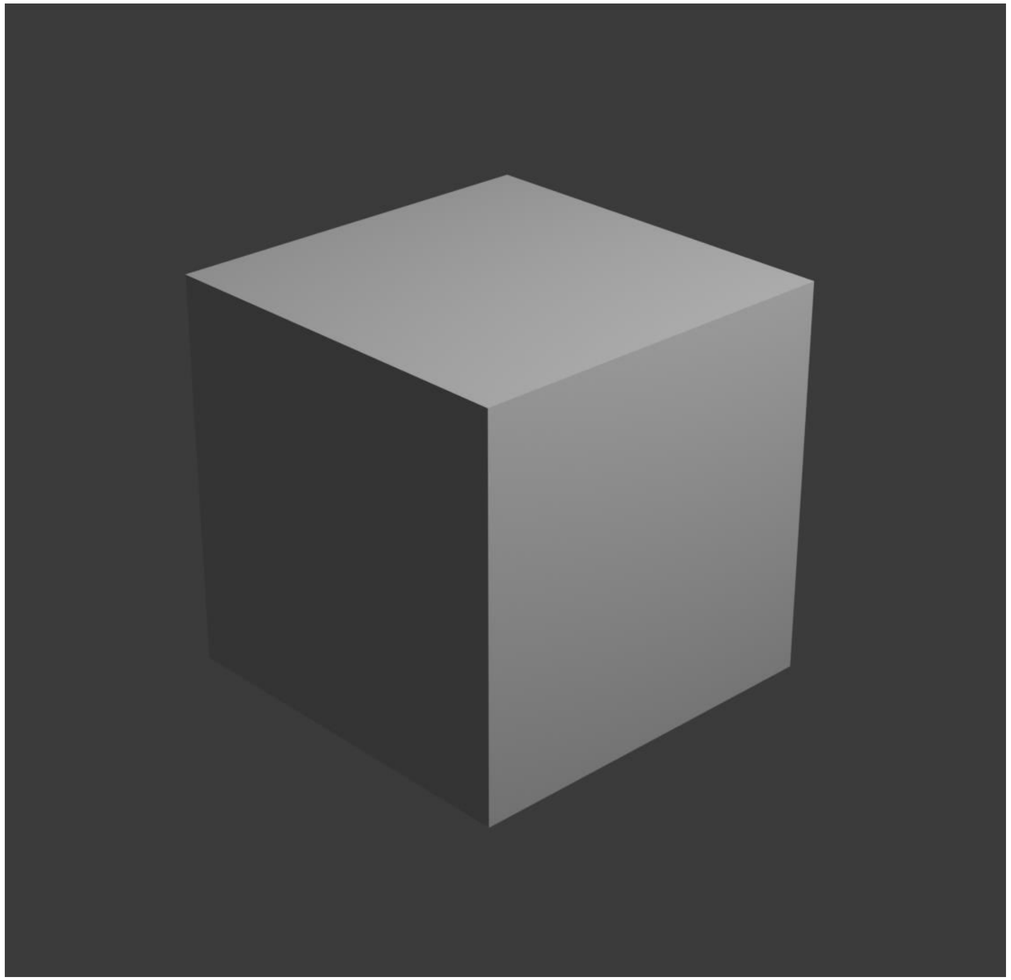
- 3D Object Files
- 3D Scene Files
- Object Space vs World Space vs Camera Space
- Camera Parameters

3D Object Files

- 3D objects, unlike most data on computers, is usually represented in two or more pieces.
  - Geometry Files (.obj, .stl, .fbx, .gitf)
  - Texture Files (images)
- You will want to make sure all pieces are present when you try to import files.

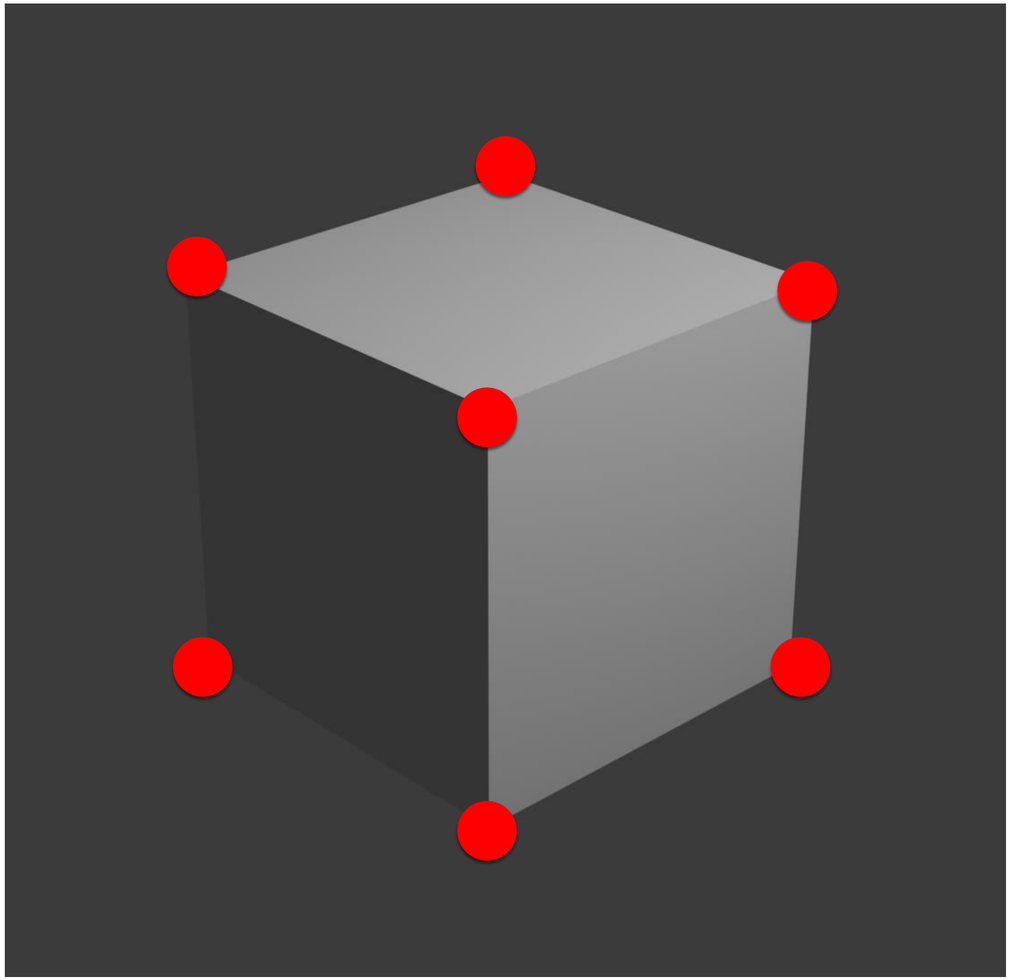
3D Geometry Files

- 3D objects are made up of
  - Vertices
  - Edges
  - Faces
- Every 3D object file must specify these in some way.



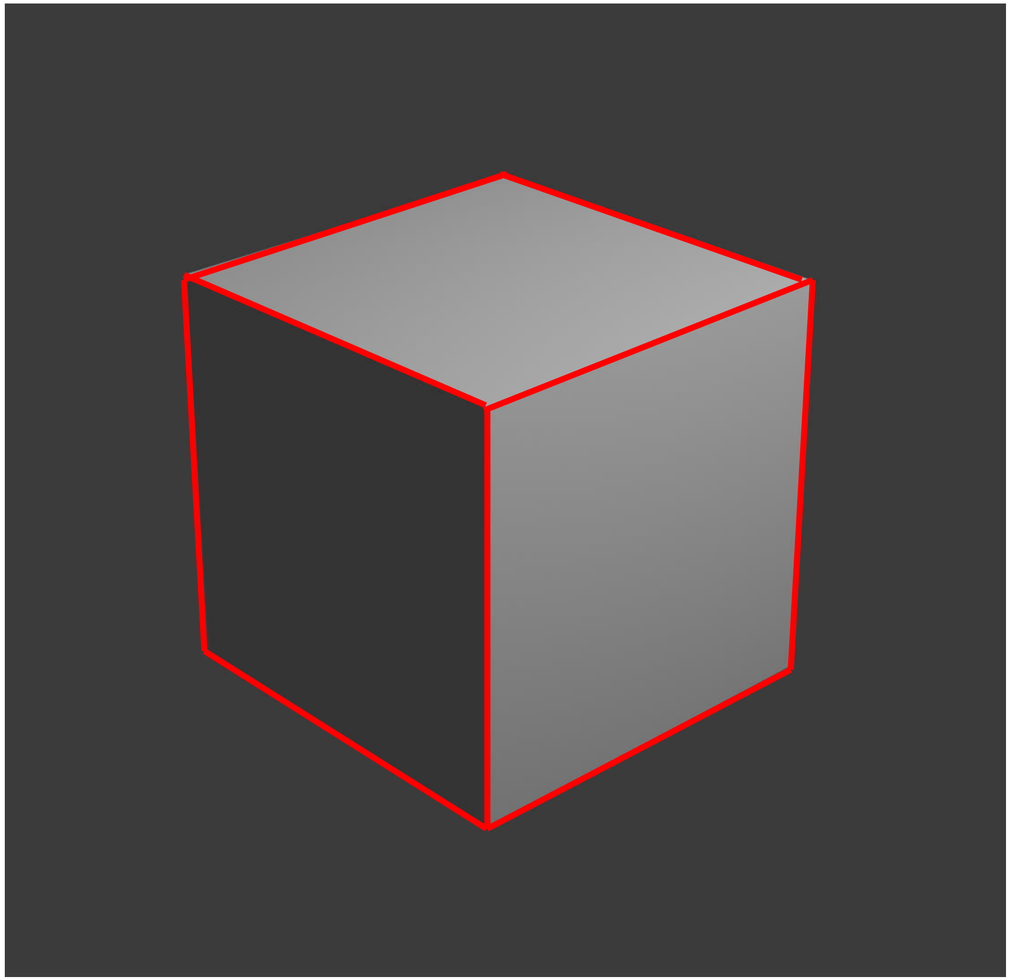
3D Geometry Files

- 3D objects are made up of
  - **Vertices**
  - Edges
  - Faces
- Every 3D object file must specify these in some way.



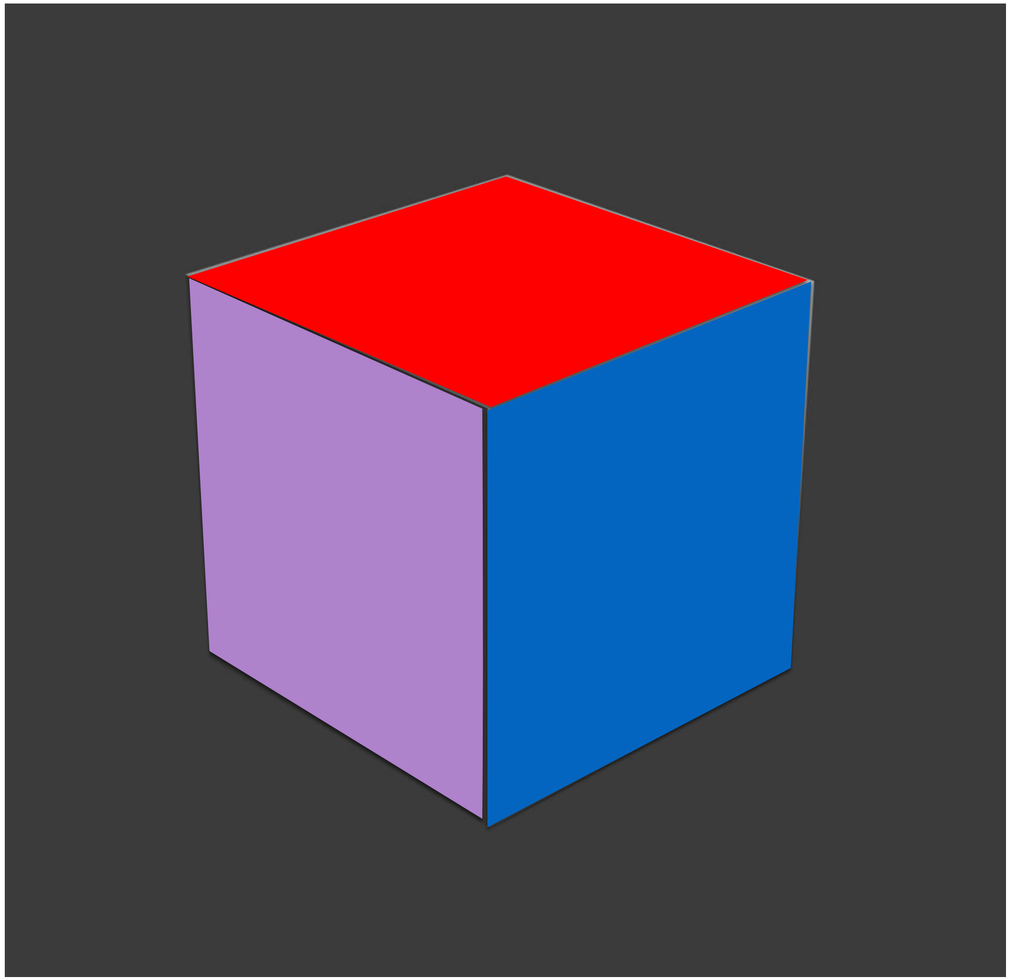
3D Geometry Files

- 3D objects are made up of
  - Vertices
  - **Edges**
  - Faces
- Every 3D object file must specify these in some way.



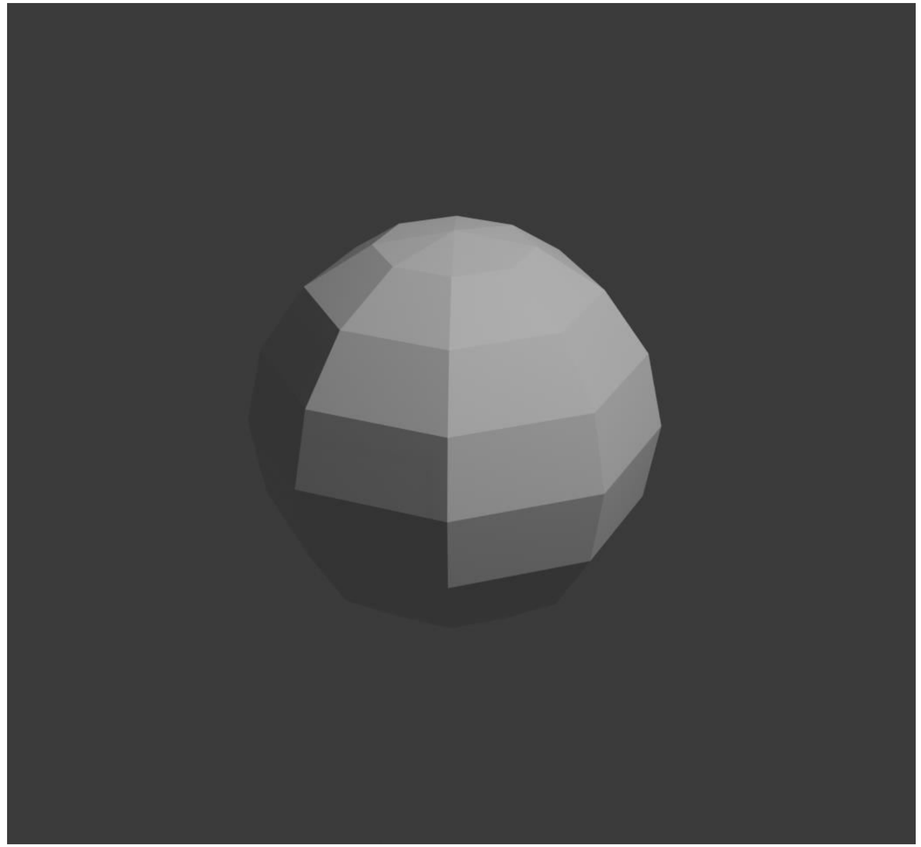
3D Geometry Files

- 3D objects are made up of
  - Vertices
  - Edges
  - **Faces**
- Every 3D object file must specify these in some way.



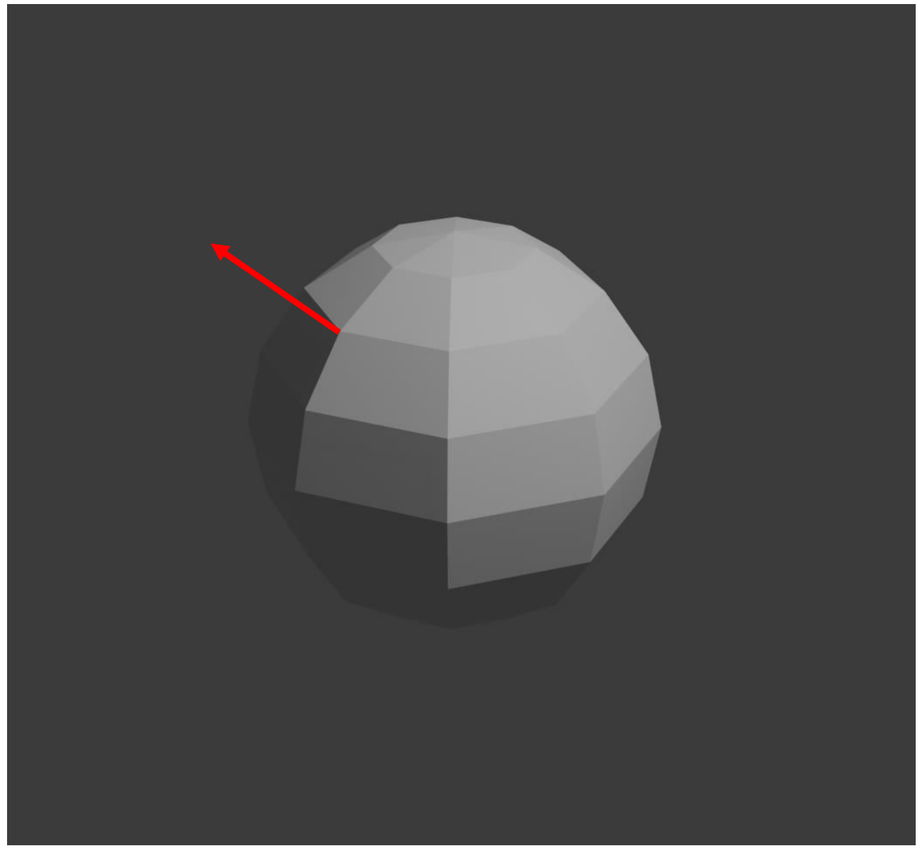
3D Geometry Files

- Object files may also specify
  - Vertex Normals
  - Face Normals
  - UV Coordinates (more on this later)



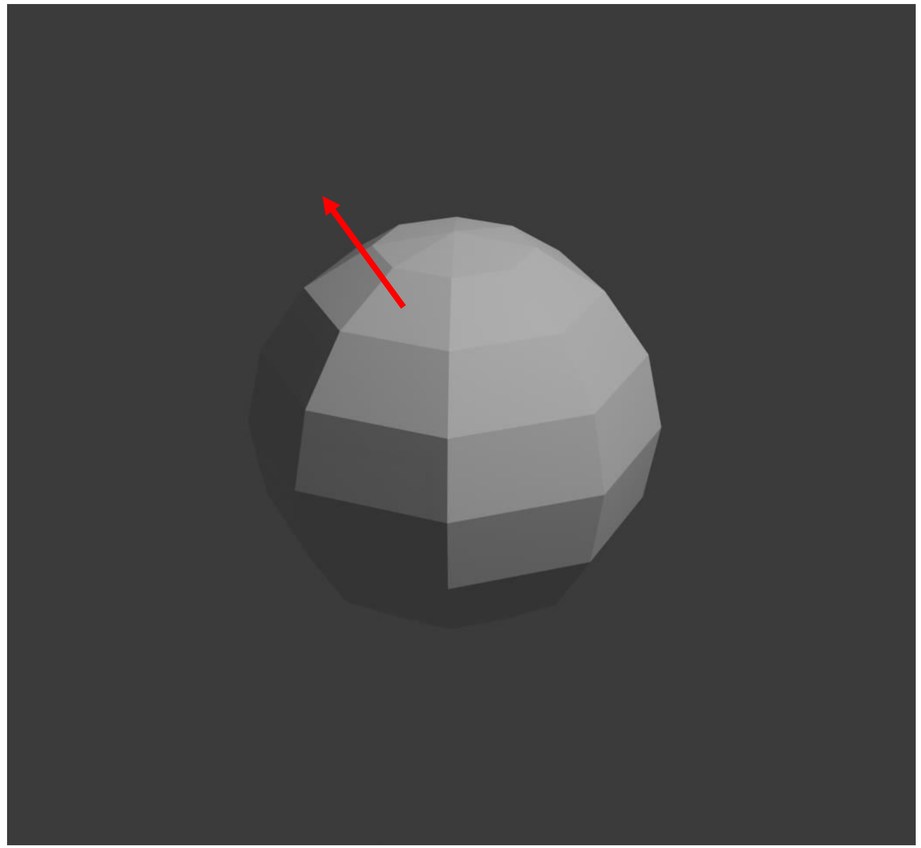
3D Geometry Files

- Object files may also specify
  - **Vertex Normals**
  - Face Normals
  - UV Coordinates (more on this later)



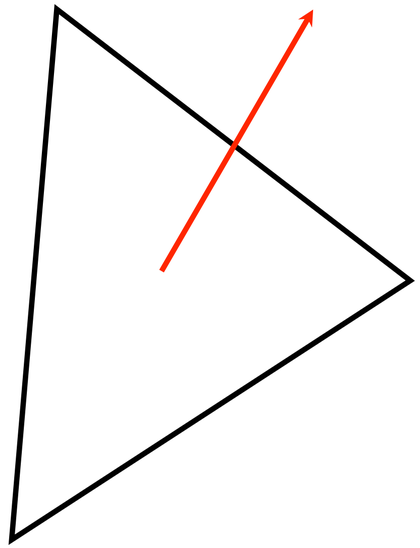
3D Geometry Files

- Object files may also specify
  - Vertex Normals
  - **Face Normals**
  - UV Coordinates (more on this later)



Normals

- **Normal Vector** – The direction orthogonal to a surface.
- It's useful to determine the *normal* to the polygonal face
  - Visibility
  - Lighting
- Be consistent—usually go with outward facing
- More on this later…



Example Geometry File (.obj)

```
v 0.317288 -0.397295 0.364448
v 0.313121 -0.404680 0.424303
v 0.289638 -0.411984 0.363044
v 0.287063 -0.417912 0.423390
v 0.345683 -0.367458 0.305622
v 0.341299 -0.375944 0.366232
v 0.317818 -0.392423 0.305485
v 0.223323 -0.431749 0.300839
v 0.251544 -0.422435 0.302332
v 0.239350 -0.431990 0.361842
v 0.262332 -0.422664 0.362137
v 0.261632 -0.427221 0.422701
v 0.274846 -0.418754 0.252658
v 0.231211 -0.430966 0.245486
```

3D Scene Files

- 3D Scene files are usually one file and contain information about the
  - Objects
  - Lighting
  - Camera
- Formats include .blend and .usd

Example 3D Scene Files

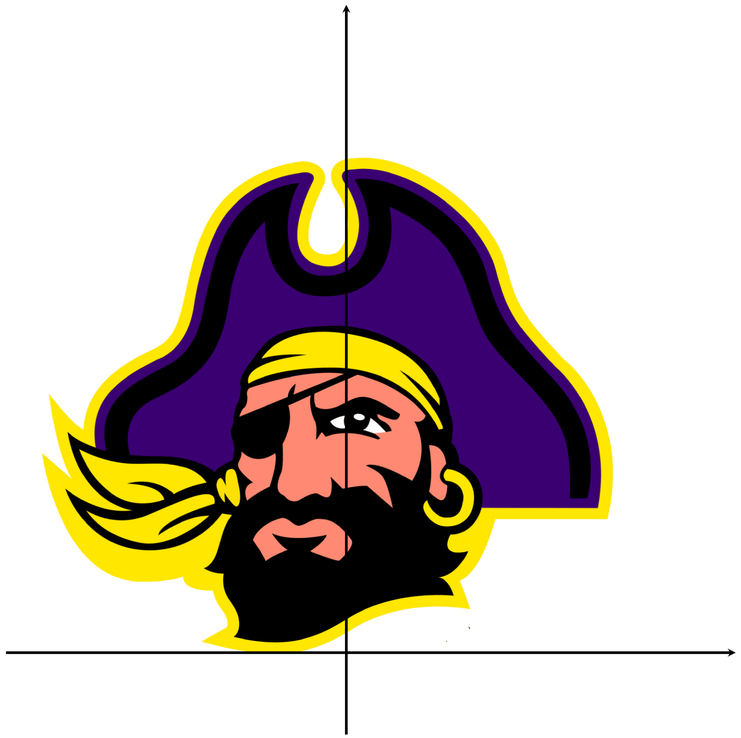
```
BACKGROUND: ( 0.10,  0.10,  0.30)
AMBIENT:    ( 0.40,  0.40,  0.40)

SPHERE_LIGHT: ( 2.00,  2.00,  2.00) 0.25  ( 1.00,  1.00,  1.00)
SPHERE_LIGHT: ( 2.00,  2.00, -2.00) 0.25  ( 1.00,  1.00,  1.00)
SPHERE_LIGHT: (-2.00,  2.00, -2.00) 0.25  ( 1.00,  1.00,  1.00)
SPHERE_LIGHT: (-2.00,  2.00,  2.00) 0.25  ( 1.00,  1.00,  1.00)

SPHERE_LIGHT: ( 8.50,  0.00,  2.00) 0.50  ( 1.30,  1.30,  1.30)
SPHERE_LIGHT: ( 8.50,  0.00, -2.00) 0.50  ( 1.30,  1.30,  1.30)

#Globe
#               Position       Radius   Diffuse Color      Specular Color    Phong  Specular Coeff.
SPHERE: ( 7.00, -2.00,  0.00) 1.00  (0.95, 0.95, 0.95) (0.95, 0.95, 0.95)   1.00    0.10  MATTE
TEXTURE: 270 135 ./textures/globe.ppm

#Alcove side walls
RECT   : Z  3.00 NEG ( 7.00, -5.00) (10.00,  2.00)  (0.30, 0.40, 0.70) (0.95, 0.95, 0.95) 20.00  0.40  MATTE
RECT   : Z -3.00 POS ( 7.00, -5.00) (10.00,  2.00)  (0.30, 0.40, 0.70) (0.95, 0.95, 0.95) 20.00  0.01  MATTE

#Alcove front wall
RECT   : X 10.00 NEG (-5.00, -3.00) ( 3.00,  3.00)  (0.10, 0.30, 0.70) (0.95, 0.95, 0.95) 20.00  0.01  MATTE
MAPPING: (10.00, -5.00,  0.00) (10.00, -5.00,  5.12) (10.00, -0.24,  0.00)
TEXTURE: 256 238 ./textures/rockface.ppm

#Alcove ceiling
RECT   : Y  2.00 NEG (-3.00,  7.00) ( 3.00, 10.00)  (0.10, 0.30, 0.70)  (0.95, 0.95, 0.95)  20.00  0.01  MATTE

#Alcove mirror.
RECT   : X  9.99 NEG (-3.00, -1.00) ( 1.00,  1.00)  (0.05, 0.05, 0.05)  (0.99, 0.99, 0.99)  20.00 0.999  MIRROR

#Alcove lamp posts
CYLINDER: ( 8.50, -5.00,  2.00) 0.10 ( 0.00, 1.00,  0.00) 4.50  (0.05, 0.05, 0.05)  (0.99, 0.99, 0.99)  20.00
     0.90

CYLINDER: ( 8.50, -5.00, -2.00) 0.10 ( 0.00, 1.00,  0.00) 4.50  (0.05, 0.05, 0.05)  (0.99, 0.99, 0.99)  20.00
     0.90
  .
  .
  .
```

Spaces

- 3D objects have multiple sets of (x,y,z) coordinates depending on whether you are referencing
  - The Object
  - The Scene
  - The Camera
- These different coordinate systems are called “spaces” and will be important to understand throughout the course.

Object Space

- The coordinate system used to define an object
- Choice of origin and coordinate axes are arbitrary, but usually chosen to make object definition the simplest
- Usually centered about (0,0,0).

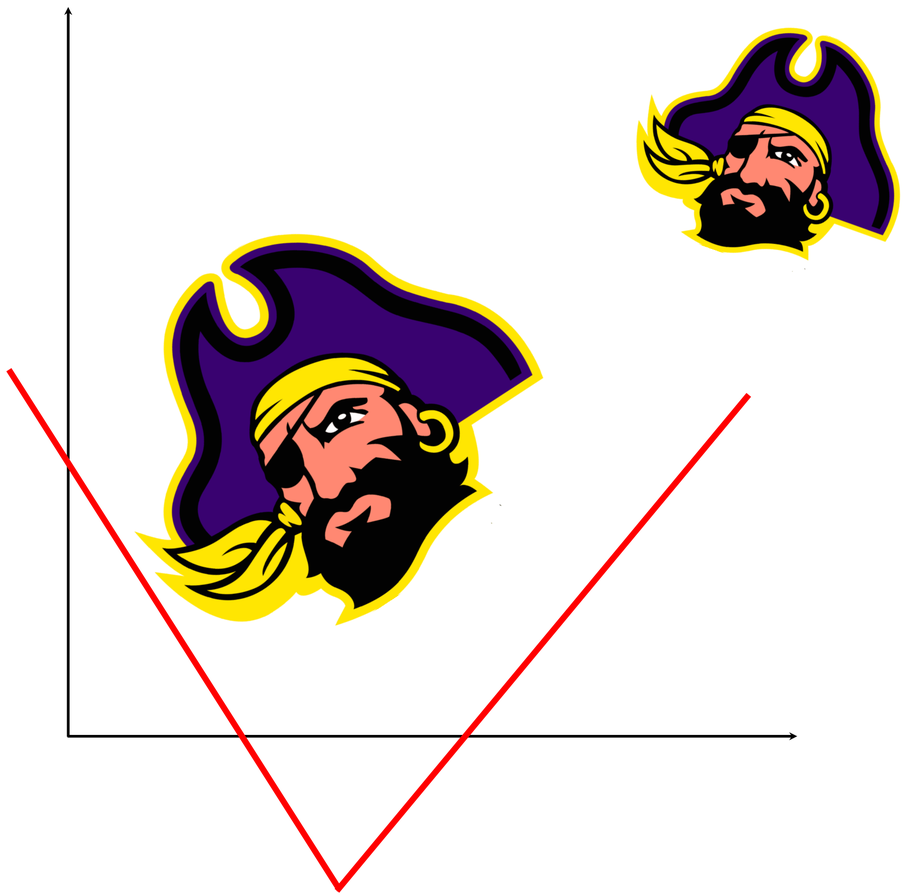


World Space

- The “world space” defines the space in which objects can live in the scene
- Determine the object’s
  - Size
  - Orientation
  - Location
- Fixed origin and axes for all objects

Camera Space

- Where is the object relative to the camera?
- Affected by the camera's location and viewing direction.

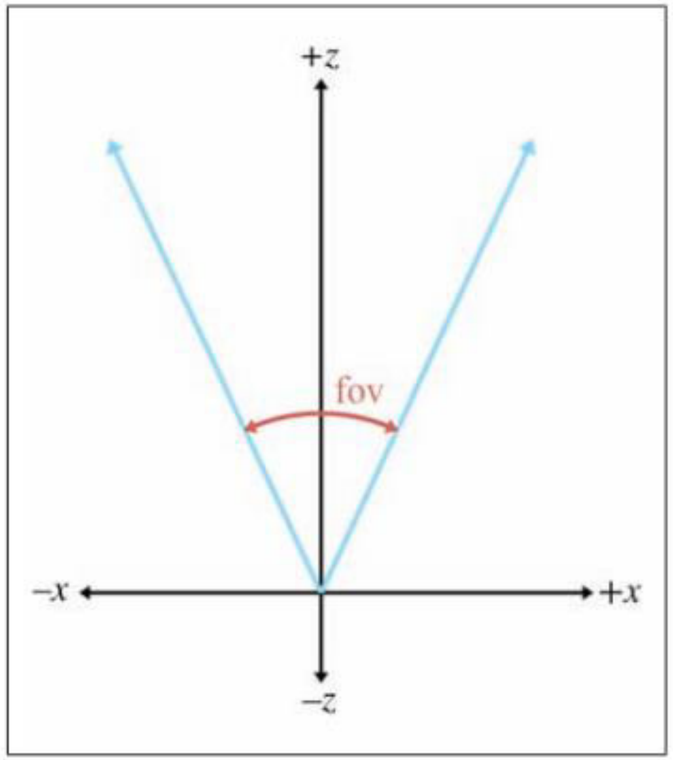


Camera Parameters

- Every camera needs four defined parameters
  - **Location**, sometimes called “View From”
  - **Viewing Direction**, sometimes called “Look At”
  - **Focal Length,** how zoomed in the camera is
  - **Aspect Ratio,** default is 16:9 for widescreen display

Field of View

- All cameras have a limited field of view
- Field of view depends on the focal length
  - Zoomed in - smaller
  - Zoomed out - larger



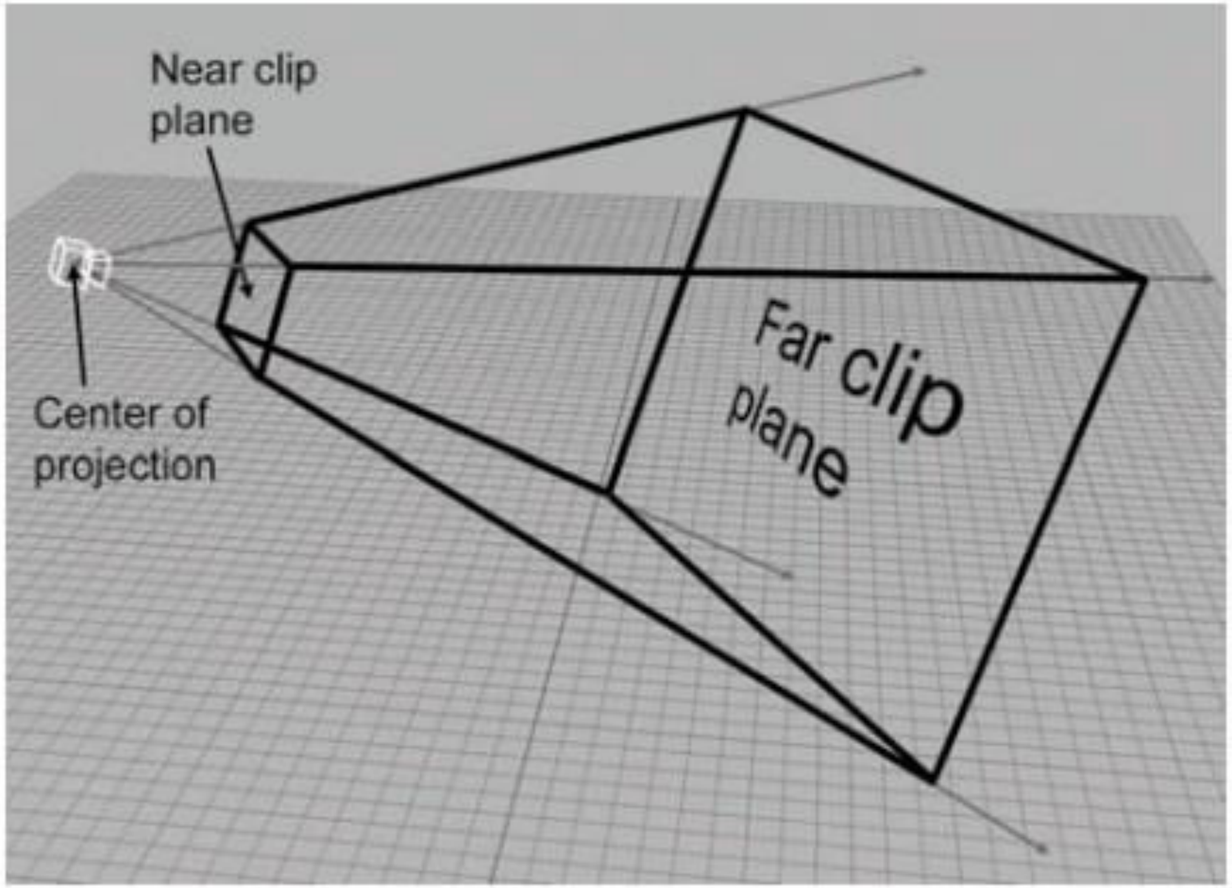
Near and Far Planes

- We don’t want to render things behind us, or perhaps even just barely in front of us
- We don’t care about things too far away to see well

“Near plane”

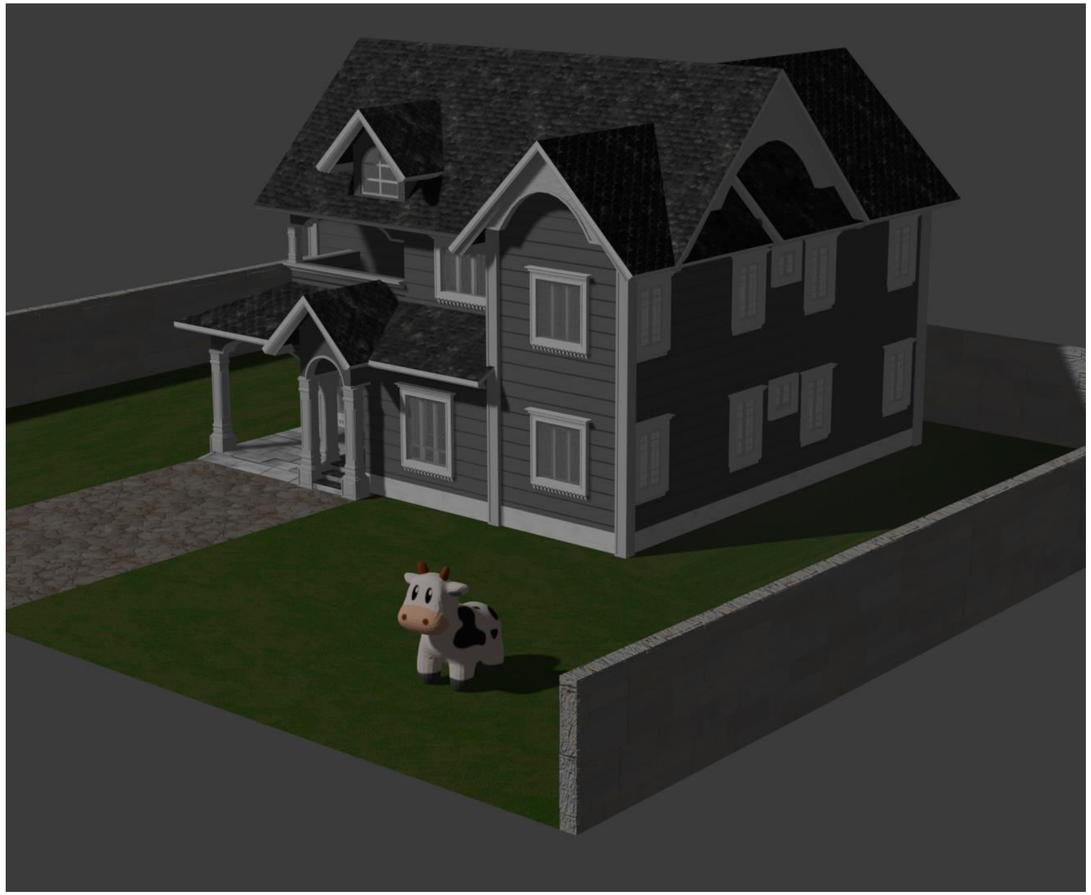
“Far plane”

View Frustum



Demo: Blender Scenes

- Importing Objects
- Translation, Rotation, Scale
- Changing the Camera



Hotkeys

- Rotate View – Middle Mouse Button
- Pan View – Shift + Middle Mouse Button
- Zoom In – Middle Mouse Wheel
- Camera View – Numpad 0
- Make Current View Camera View – Ctrl + Alt + Numpad 0
- Render Scene – F12

Hotkeys

- Translation – **g**
- Rotation – **r**
- Scale – **s**

- Move, rotate, scale only in x-axis – **g**/**r**/**s** + **x**
- Move, rotate, scale only in y-axis – **g**/**r**/**s** + **y**
- Move, rotate, scale only in z-axis – **g**/**r**/**s** + **z**
- Move, scale only in xy plane – **g**/**s** + Shift-**z**
- Move, scale only in yz plane – **g**/**s** + Shift-**x**
- Move, scale only in xz plane – **g**/**s** + Shift-**y**

Poly Haven

- polyhaven.com
- Public Domain 3D Files
- Models that can be imported into Blender
- Textures and HDRIs that we will use later

3D Modeling Basics

CSCI 3800/6820: Computer Graphics

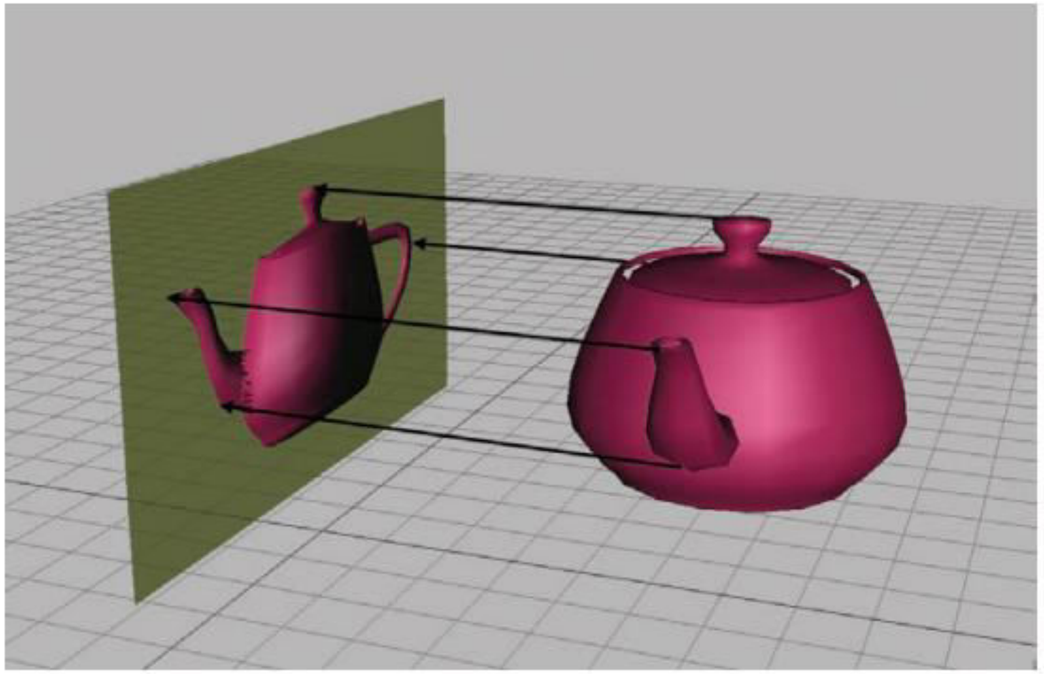
Outline

- Orthographic vs Perspective Projection
- Modeling Tools
  - Extrude, Insets, and Bevels
  - Continuous Selections and Loop Cuts
- Modifiers
  - Subsurface
  - Boolean

Projection

- To get 2D pictures of a 3D world, you have to use *projection*
  - Orthographic
  - Perspective

Orthographic Projection

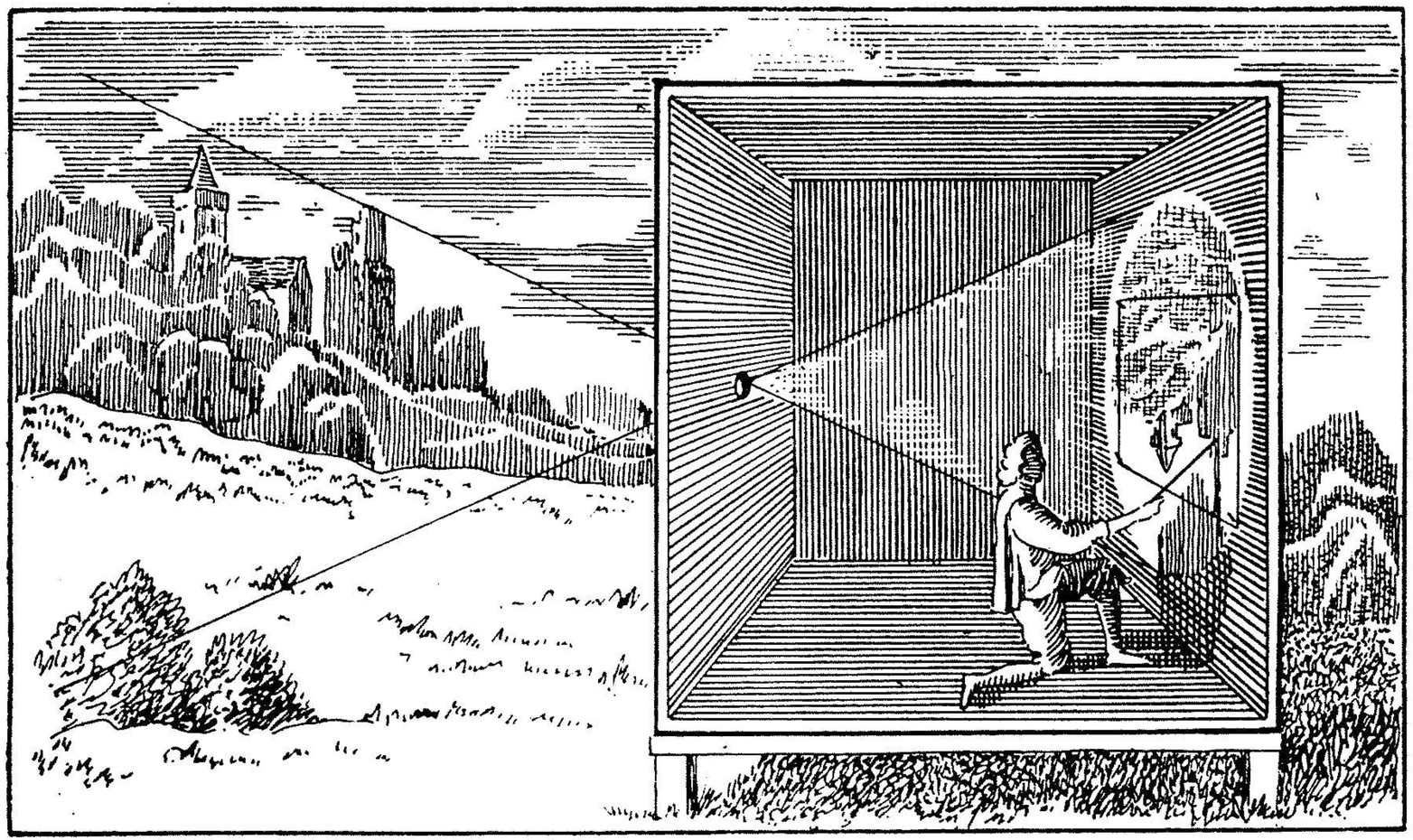


Orthographic projection is just dropping a dimension.

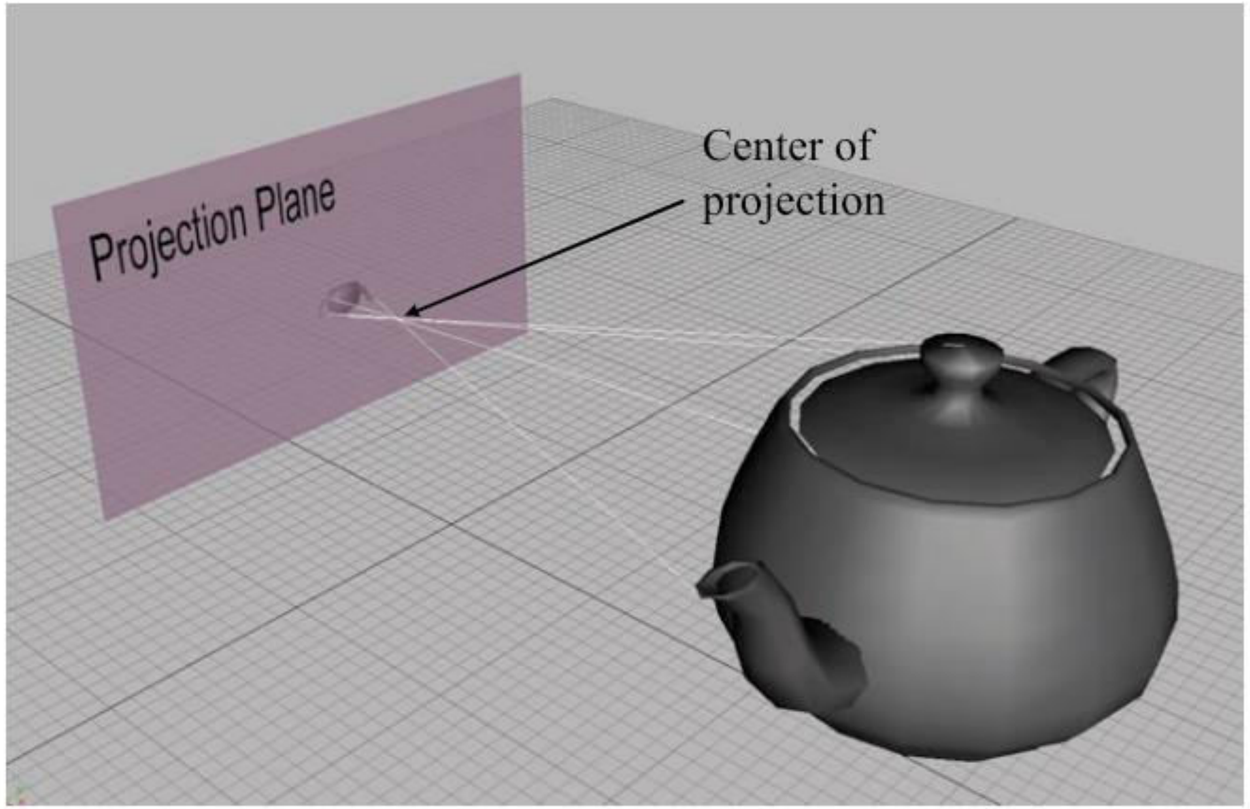
Size does not change with distance.

Perspective Projection

Camera Obscura



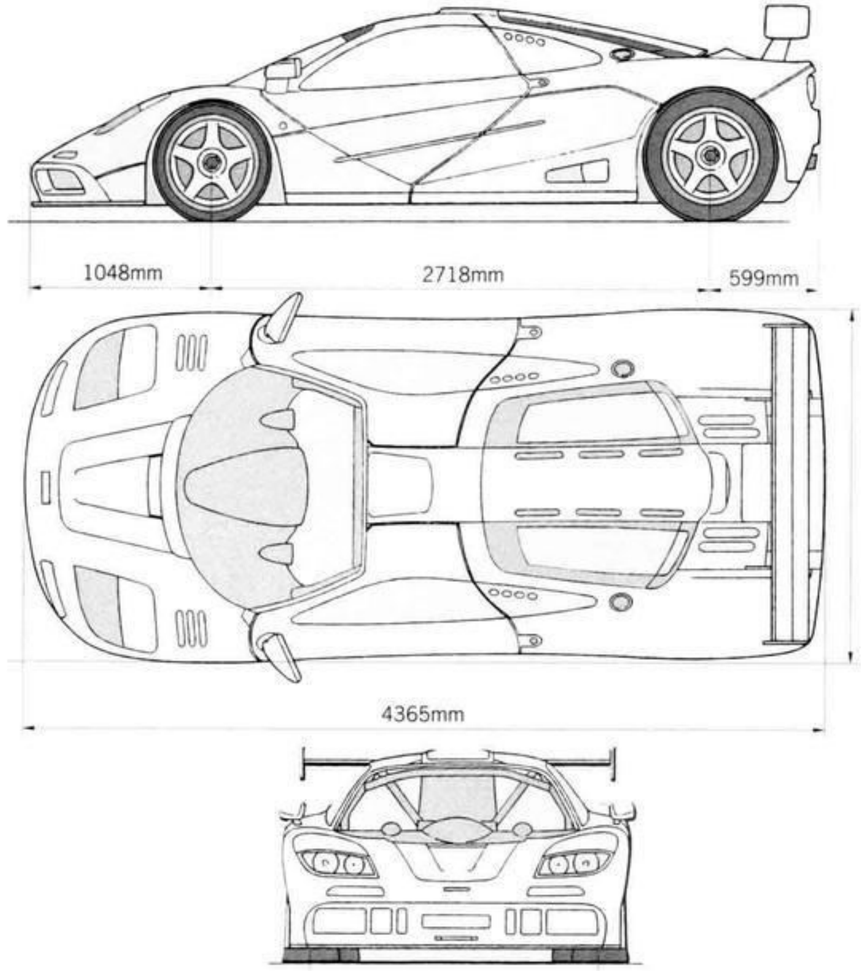
Perspective Projection



Objects appear larger/smaller based on how far away they are from the camera

Projection and Modeling

- It is usually best to model 3D objects in an orthographic projection.
- This helps prevent visual artifacts of geometry that looks good from one perspective, but not another.
- Many technical drawings use three orthographic projections.

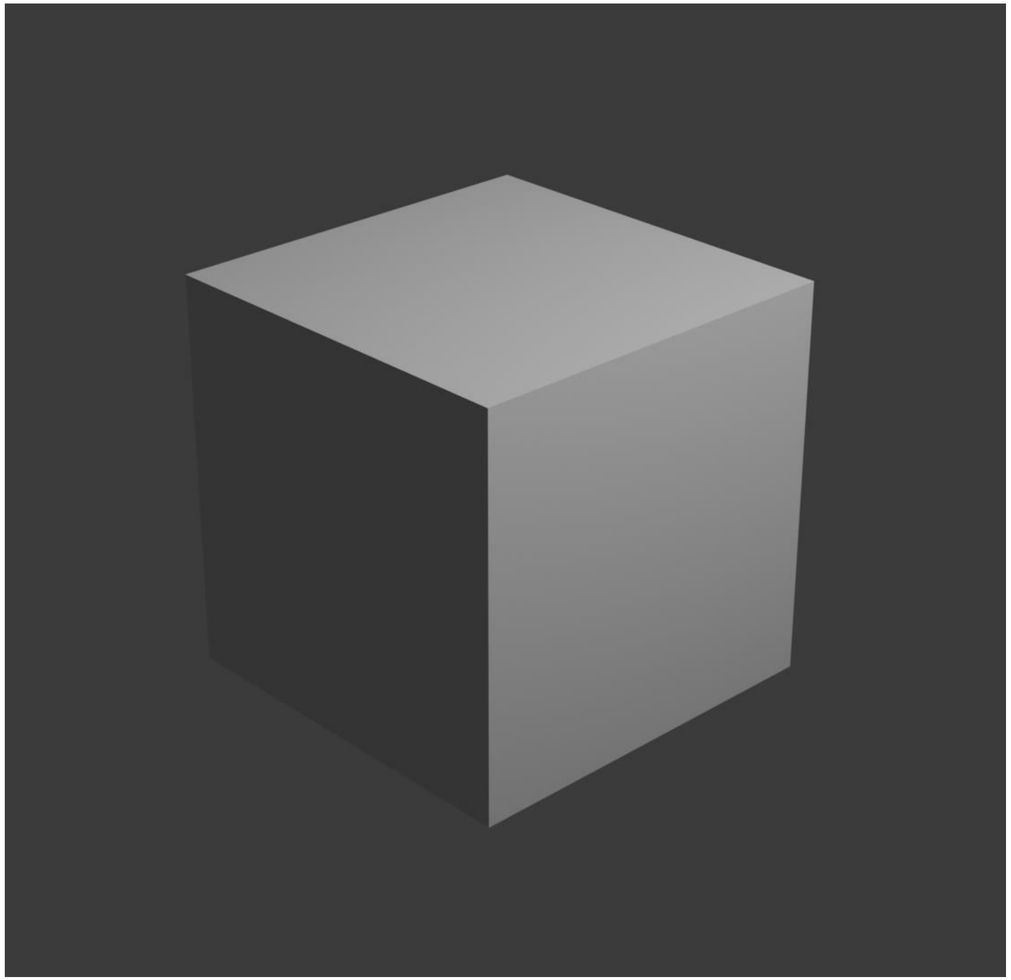


https://grabcad.com/library/car-blocks-blueprints

Let’s try it in Blender!

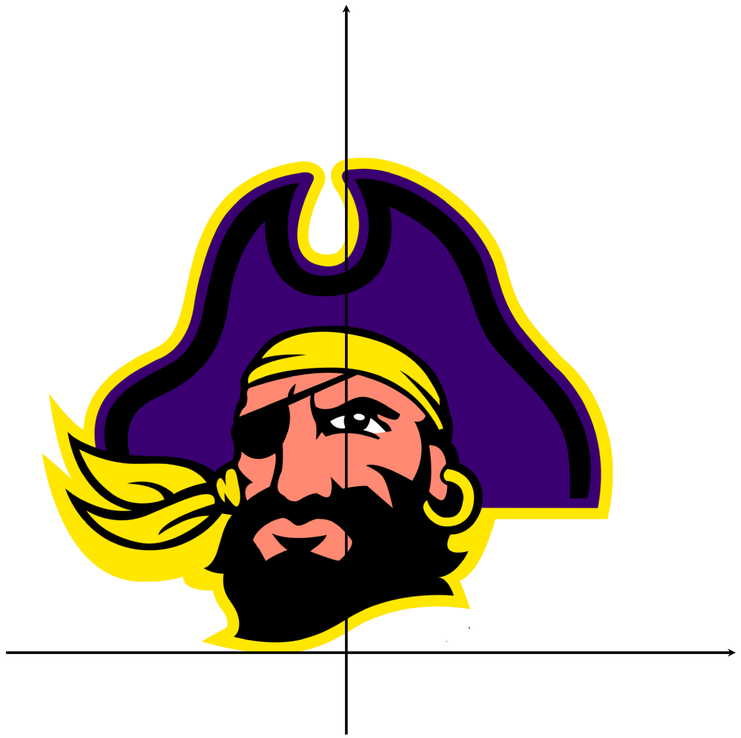
3D Modeling Basics

- The core of 3D modeling is to manipulate the object's
  - Vertices
  - Edges
  - Faces
- This can be done through translation, rotation, and scaling.



Model in Object Space

- It is best to model the object about a common center or point, usually (0,0,0).
- It is usually not recommended to model an object around where it is placed in the world, since this could cause issues if it is imported into other scenes.



Other Modeling Tools

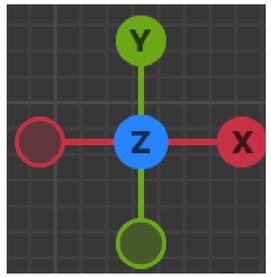
- It can be tricky knowing exactly how to move vertices, edges, and faces to create the exact 3D object shape you want.
- Most of 3D modeling tools are built to make it more intuitive to move object surfaces and add vertices where needed.
- We will spend some class time demonstrating the tools you will use on a regular basis when editing 3D objects.

Demo: Coffee Cup

- Mirror Edits
- Proportional Edits
- Continuous Selection
- Insets
- Extrude
- Loop Cuts

Hotkeys

- Orthographic Projection Modes - Numpad 1-9 or



- Edit Mode – Tab key
- Add Object – Shift+A
- Proportional Editing – O (while in edit mode)
- Change Proportional Edit Influence – Mouse Scroll Wheel
- Continuous Select – Ctrl+Left Click

Hotkeys

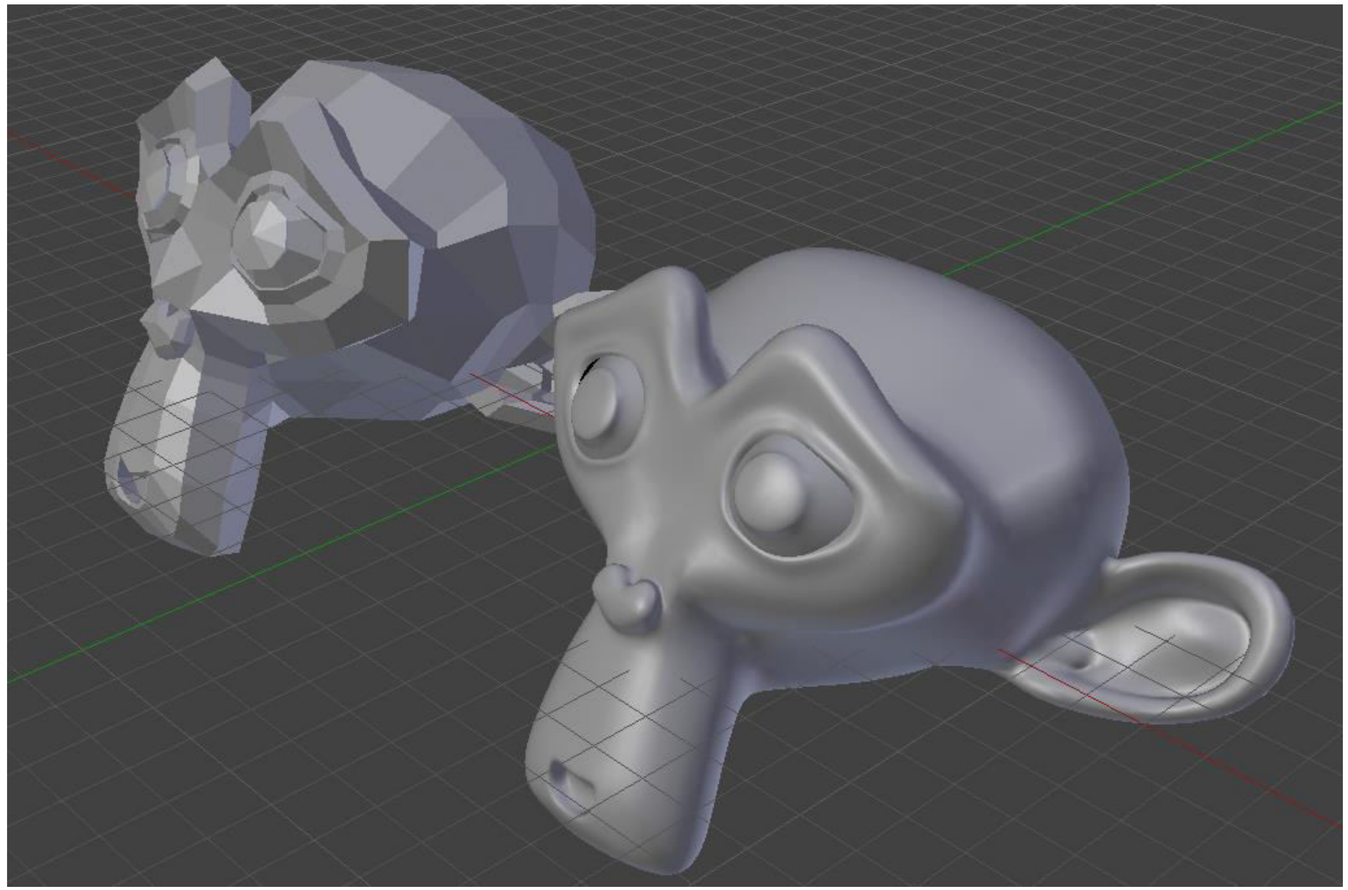
- Translate Vertex, Edge, Face - G
- Rotate Edge or Face - R
- Scale Edge or Face - S
- Inset points - I
- Extrude Face - E
- Continuous Select – Ctrl+Left Click

Modifiers

- Modeling a 3D object exactly can be difficult and time-consuming
- There are post-processing tools that Blender can use to change the geometry you have provided it.
- These tool are called **modifiers** and can simplify your workflow.
- Two common ones are the “Subsurf” and Boolean modifiers.

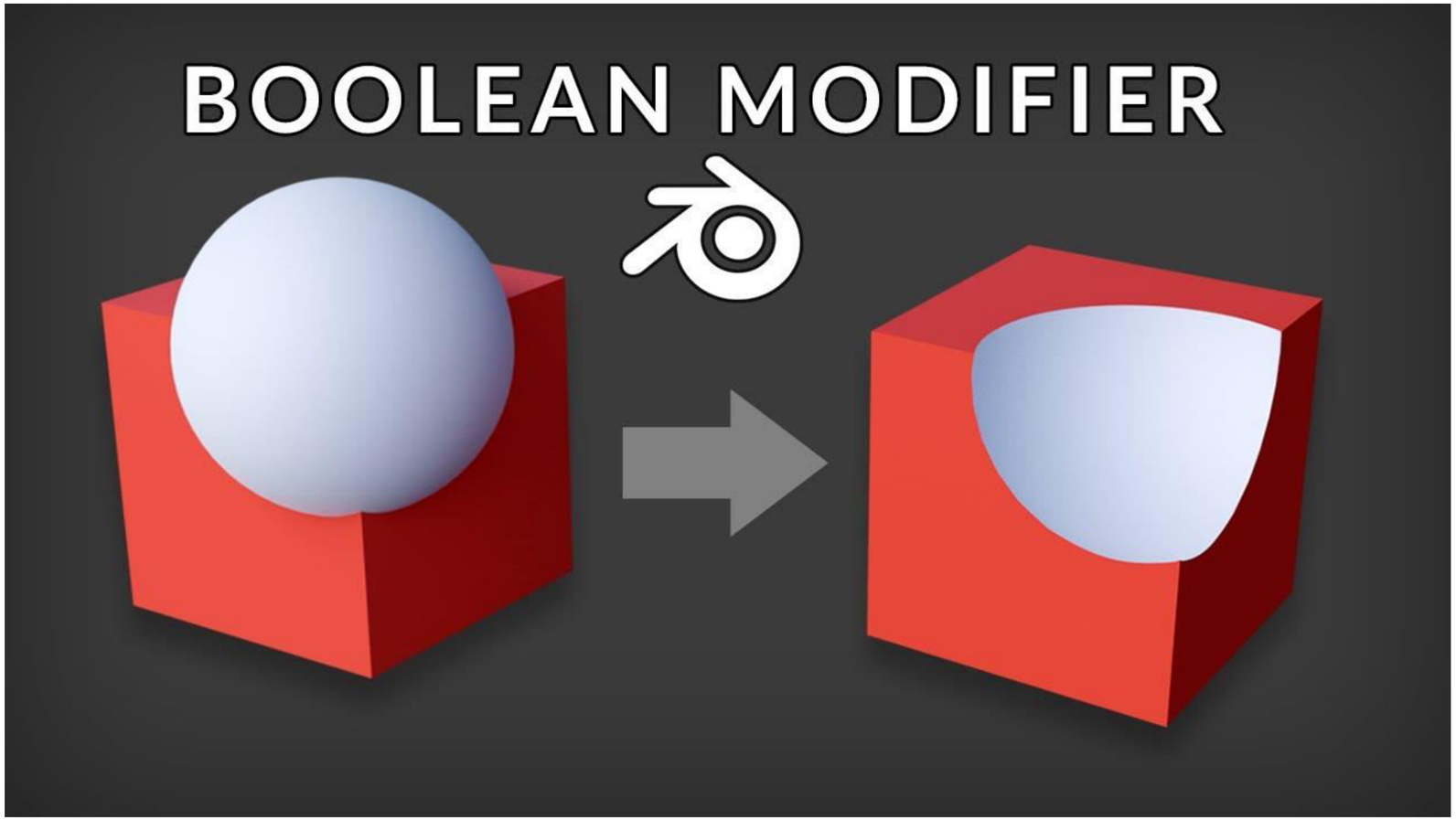
Subdivision Surface Modifier

- Generates a smooth surface between current polygonal faces on the mesh.
- Takes a coarse model to a fine-detailed model.



https://exnihilodigital.com/tutorials/subsurface-modifier/

Boolean Modifiers



- Allows you to make complex shapes by adding or subtracting two or more basic shapes.
- **Warning:** Doesn't always create consistent geometry.

https://www.youtube.com/watch?v=SMIIZddICDg

Sculpting

CSCI 3800/6820: Computer Graphics

Outline

- Sculpting Basics
  - Draw and Inflate
  - Elastic Deform and Pose
    - ARAP (As Rigid As Possible)
- Dynamic Topology
- Remeshing

What is Sculpting?

- Sculpting allows you to manipulate 3D models with more intuitive controls for making complex 3D geometries.

- Designed to feel like physically molding something out of clay.

Sculpting Tools

- Draw – Pushes the mesh in and out at the given location.
    - The most commonly used sculpting tool.

- Inflate – Similar to draw, but bubbles at the given location.
    - Helpful for quickly adding a lot of volume to the mesh.

Sculpting Tools

- Elastic Deform – Translates a chunk of the mesh to new location.

- Pose – Rotates a chunk of the mesh at a given location.

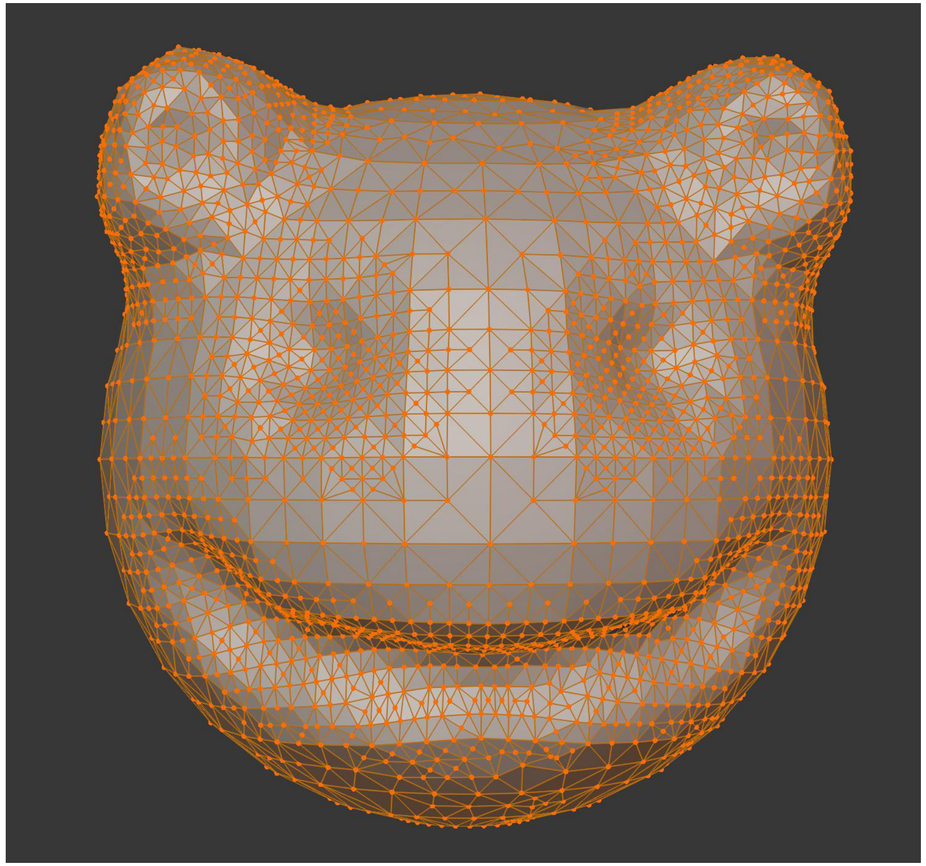
- And much much more…

Brush Options

- **Add vs Subtract –** Do you want to push the mesh in or out?
- **Strength** – How much do you want to push/pull on the mesh?
- **Radius** – How far out do you want this effect to take place?

Dynamic Topology

- When sculpting, it is important to enable Dynamic Topology (Dyntopo).
- It allows the program to add more vertices in areas of more detail.

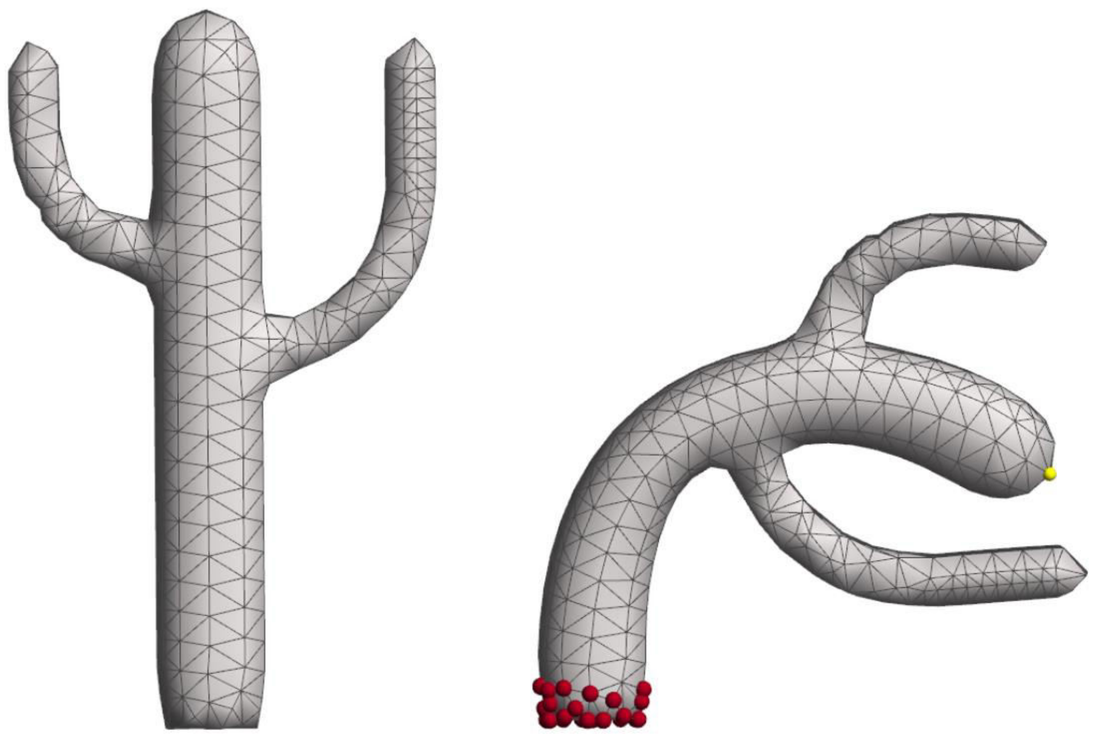


Demo: Teddy Bear

- Dynamic Topology (Dyntopo)
- Draw and Inflate
- Strength, Radius, Direction
- Elastic Deform and Pose

As Rigid As Possible (ARAP)

- ARAP is a common optimization used for editing meshes in 3D graphics.
- Optimization: If I move a vertex to a new location, what deformation
  - Minimizes total movement of all vertices?
  - Minimizes changes in distances between connected vertices?

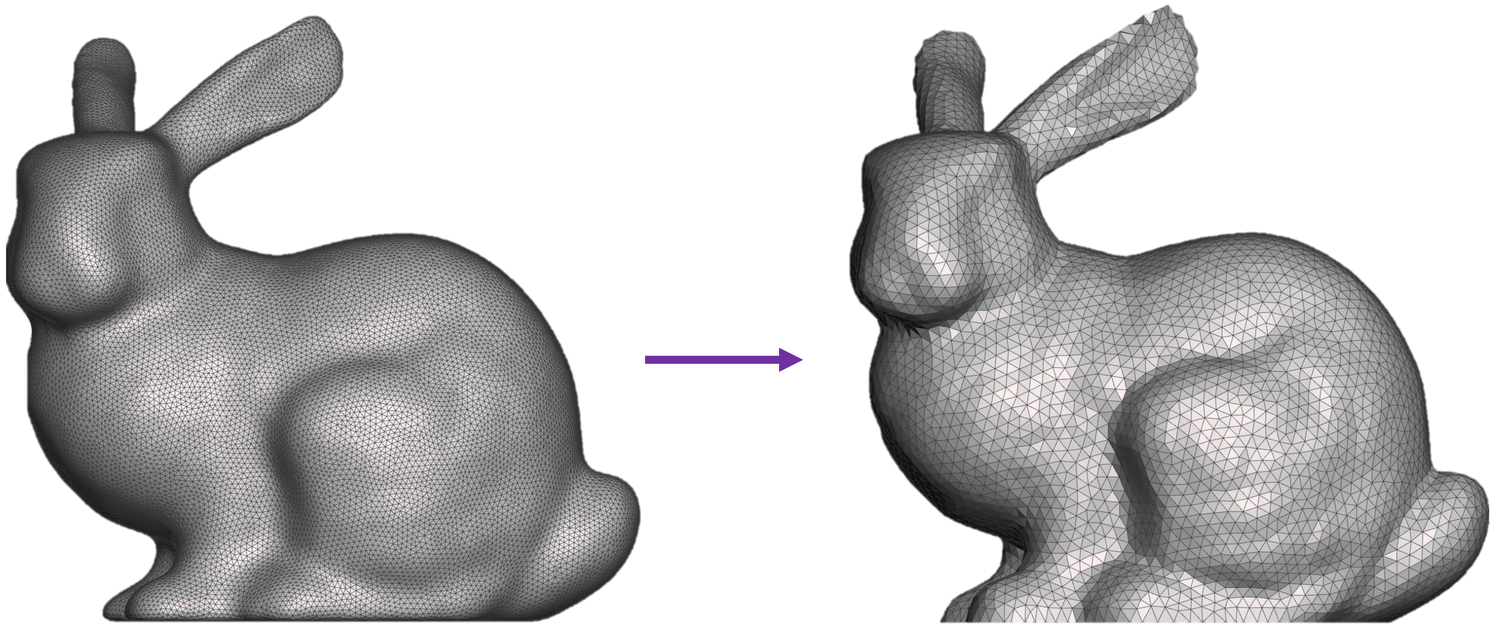


https://www.youtube.com/watch?v=ItX-qUjbkdc

Remeshing

- After a long time of sculpting, Dyntopo may add more vertices than is necessary to represent the object.

- To reduce memory and rendering time, it can be helpful to recalculate the model mesh using the remeshing tool.



https://www.gradientspace.com/tutorials/2018/7/5/remeshing-and-constraints

Sculpting Quick Tips

- Always enable Dyntopo!
- Use mirror tools when applicable
- Use Ctrl to quickly switch from adding to subtracting brushes
- Never pose until you have completed the sculpting
- For complex meshes, remesh after completion

UV Editing

CSCI 3800/6820: Computer Graphics

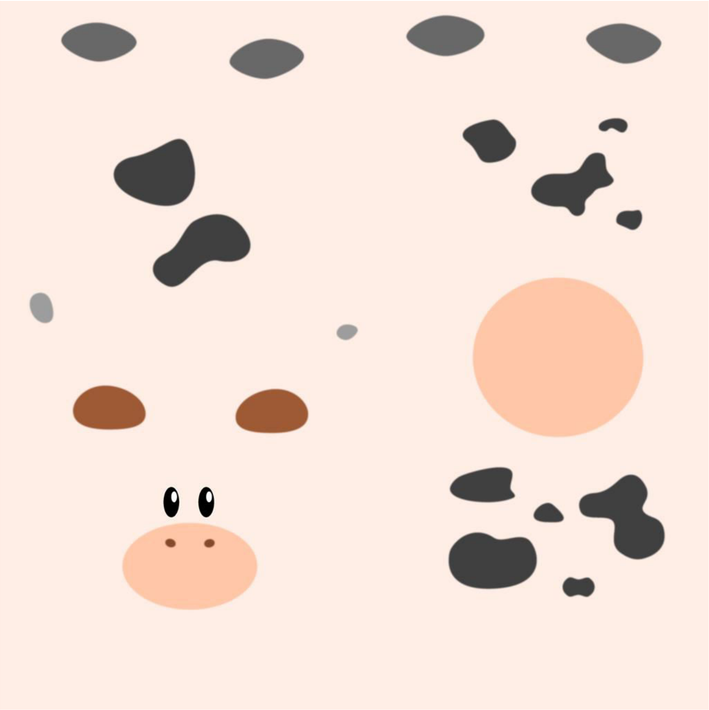
Some slides are from
Professor Parris Egbert (BYU)

Outline

- Texture Maps
- UV-Coordinates
  - Unwrapping
- Barycentric Coordinates

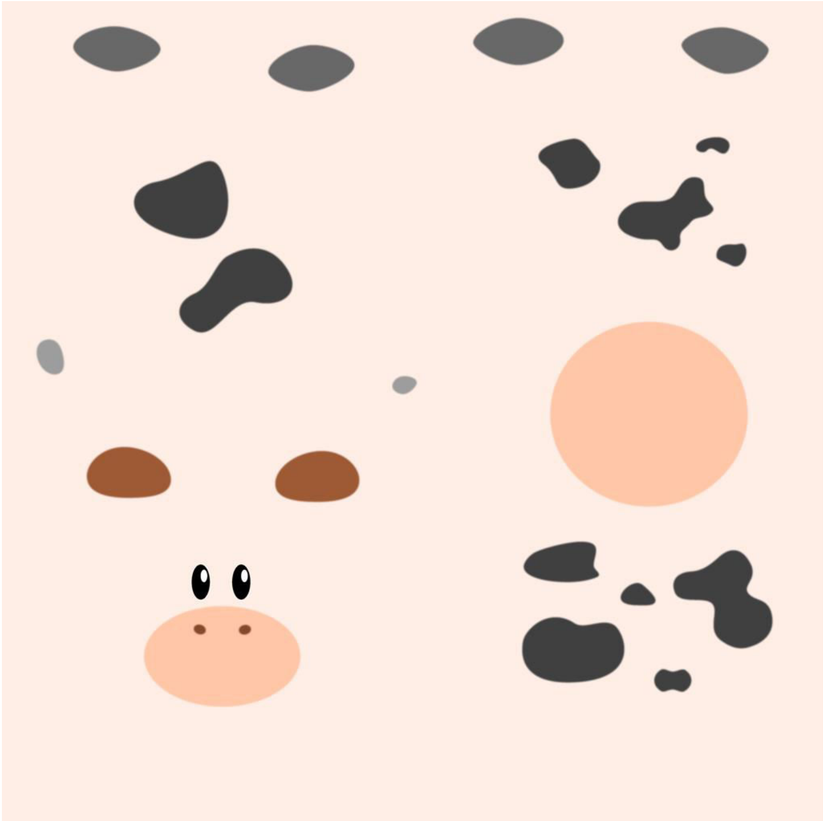
Texture Maps

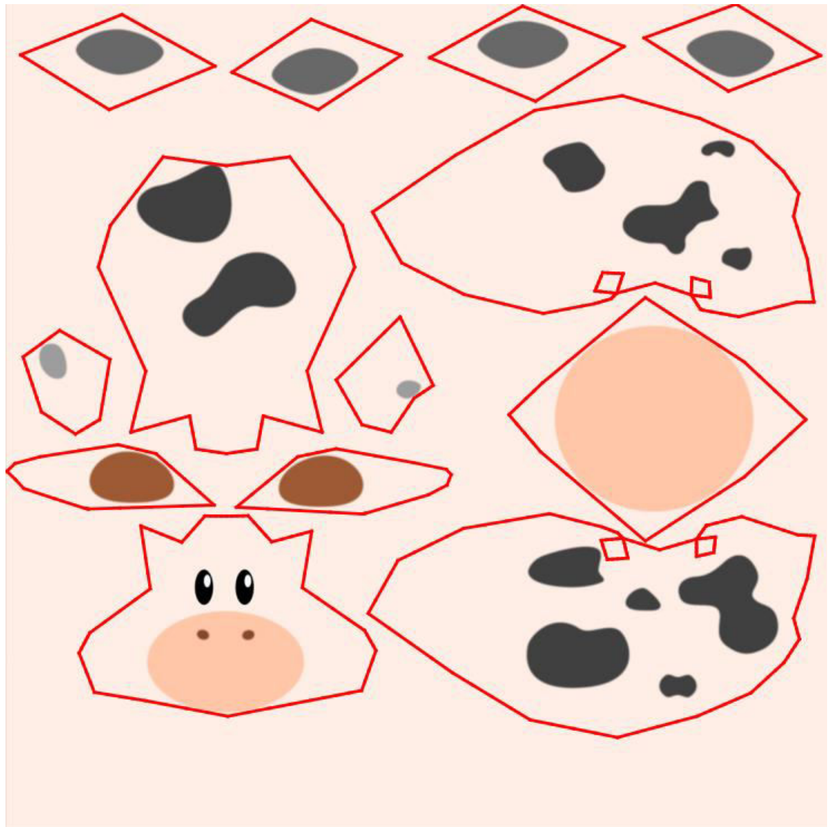
- An image that defines the colors of a 3D model.
- You may also have texture maps giving information about surface properties such as roughness and reflectance (more on this next time…)



Texture Maps

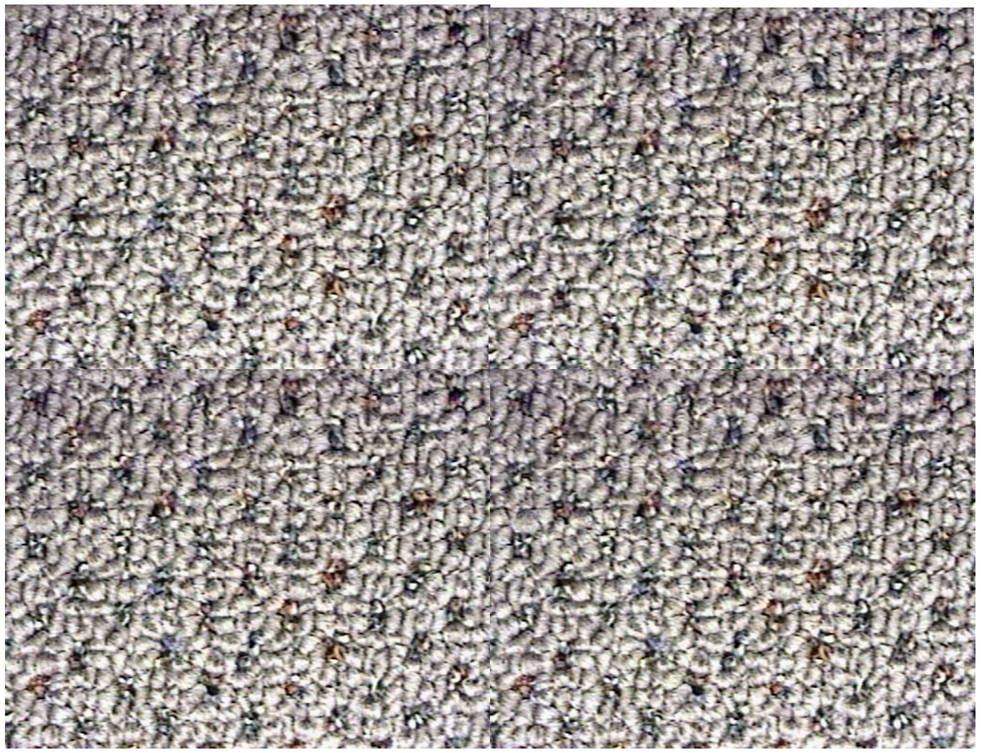
- Note: Not all parts of a texture map must be used.

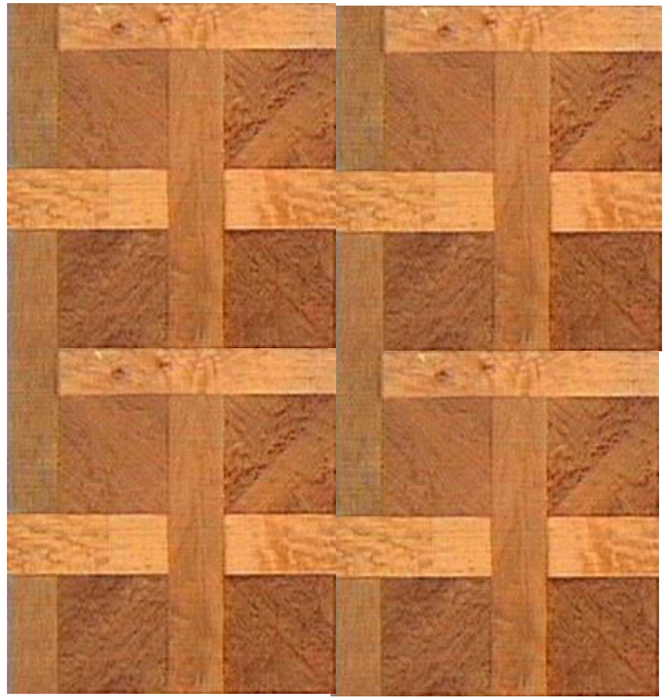




Replicating Textures

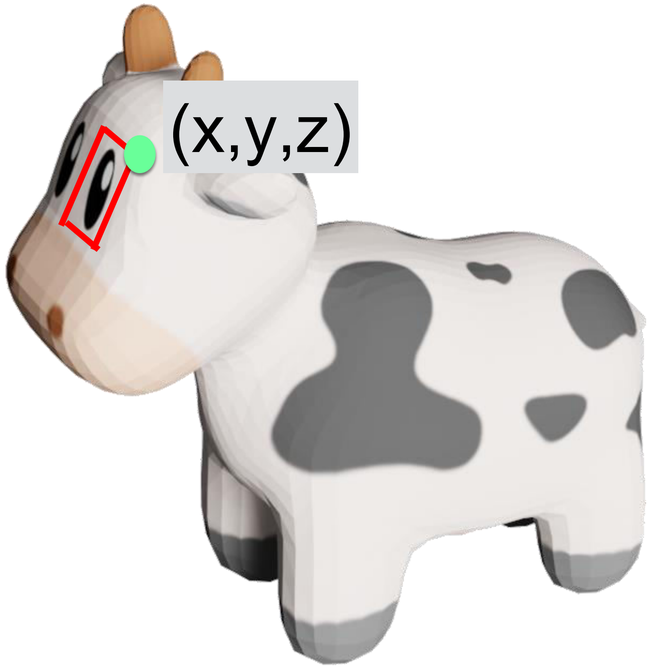
- Textures with repeating patterns can be replicated to color large areas such as floors or walls.

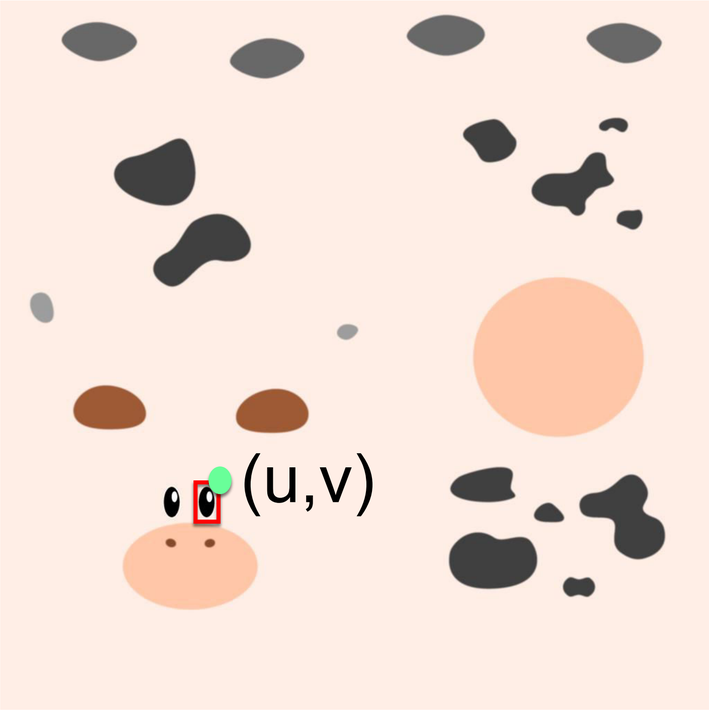




UV-Coordinates

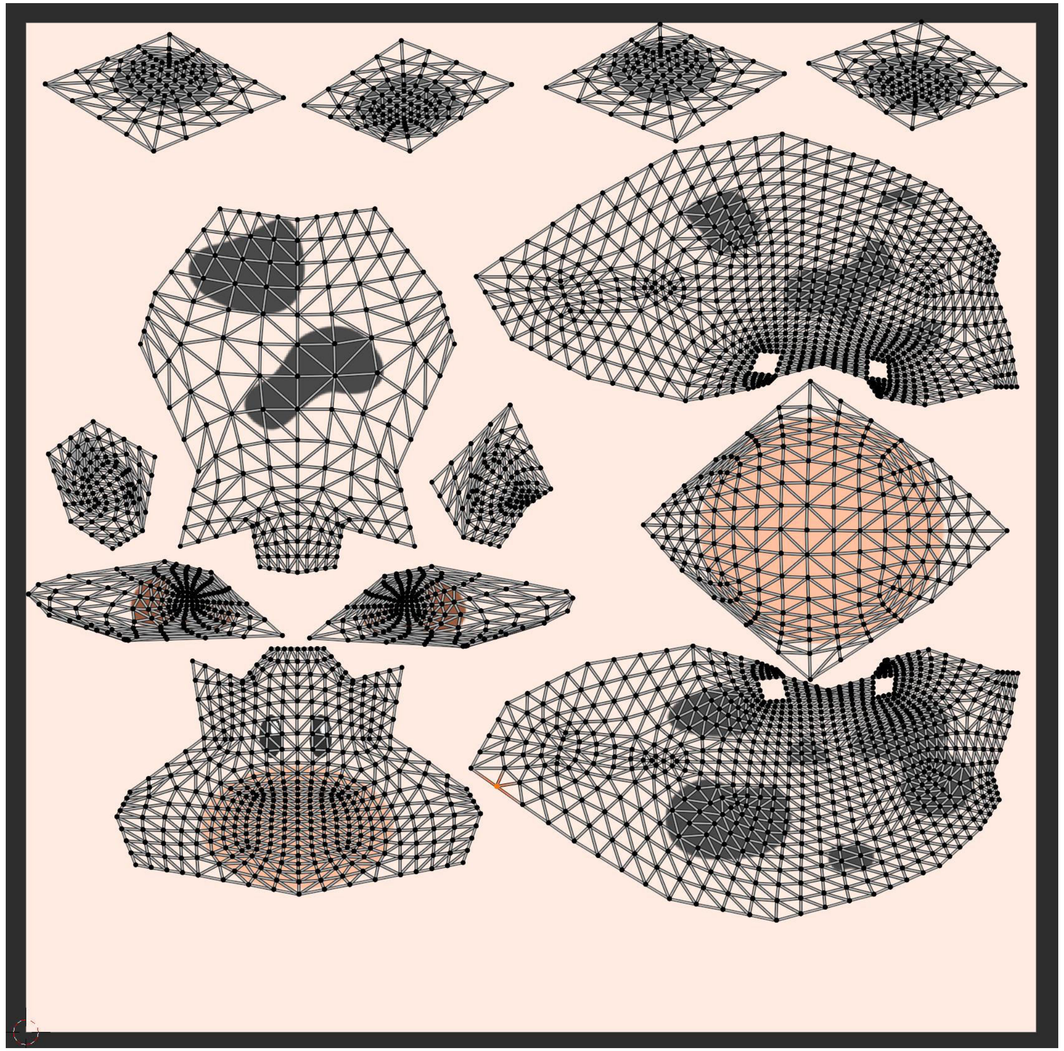
- Every vertex (x,y,z) coordinate on the 3D model needs an associated (x,y) location on the texture map.
- To avoid confusion, we call this the (u,v) location on the texture map.
- Nearby (x,y,z) locations on the 3D model do not need to be nearby (u,v) locations on the texture map.





Example UV Unwrapping

(0, 1) (1, 1)

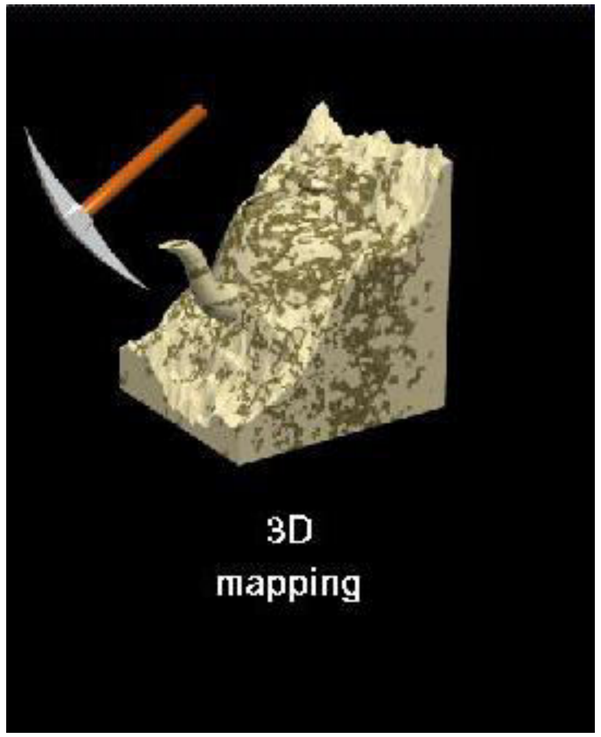


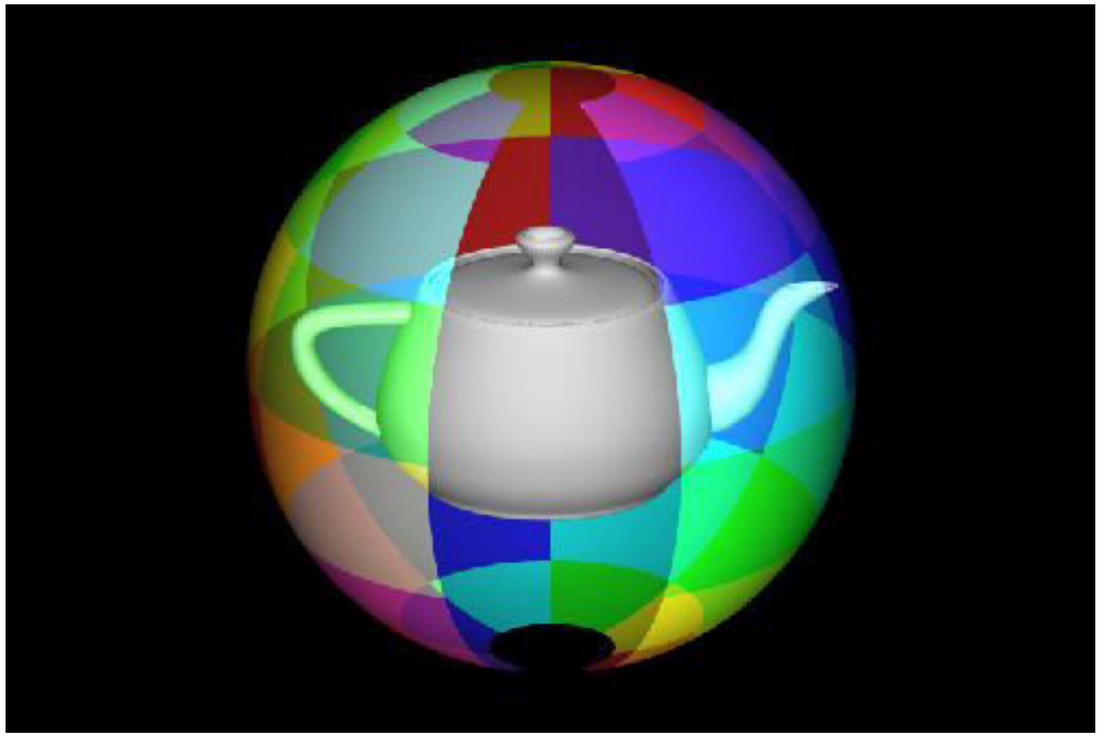
(0, 0) (1, 0)

uv-space is usually defined from 0 to 1, horizontal coordinate first

Aside: Alternative Approaches

- In addition to UV-Coordinates, there are alternative mapping techniques, such as 3D texture maps and spherical maps, be these have become less used in recent years.





Demo: Crayon Box

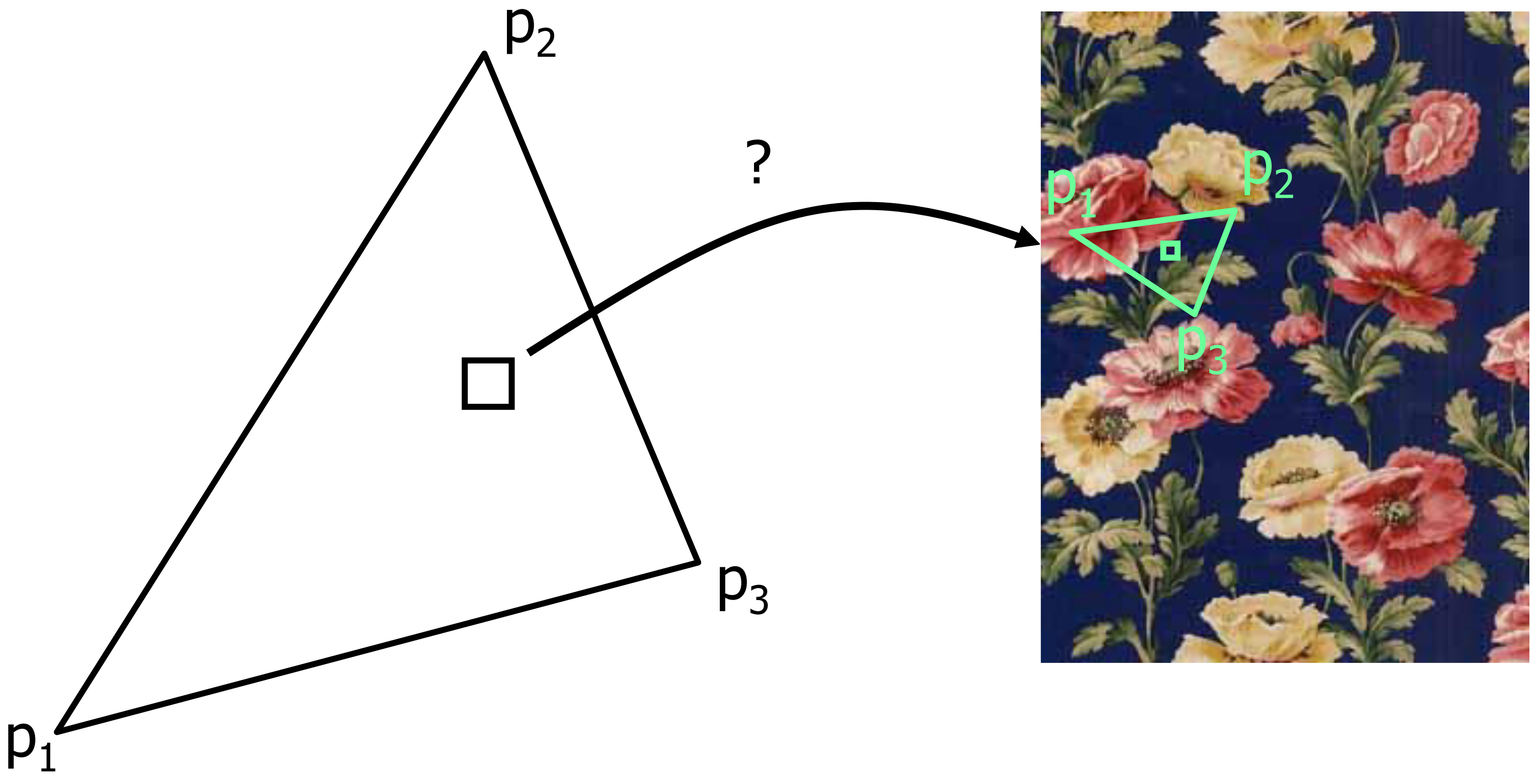
- Loading Texture Maps
- Smart Unwrapping
- Adjusting UV Coordinates

Quick Tips

- You must add a material to an object before you can do UV editing.
- Add the texture map image as “Image Texture” under “Base Color” of the material properties. You may need to reload the image in the UV edit tab too.
- In the UV Editing tab, make sure you are in Edit Mode, then press A to select all vertices and U to complete the Smart Unwrapping
- Don’t assume faces are right side up on the texture map!

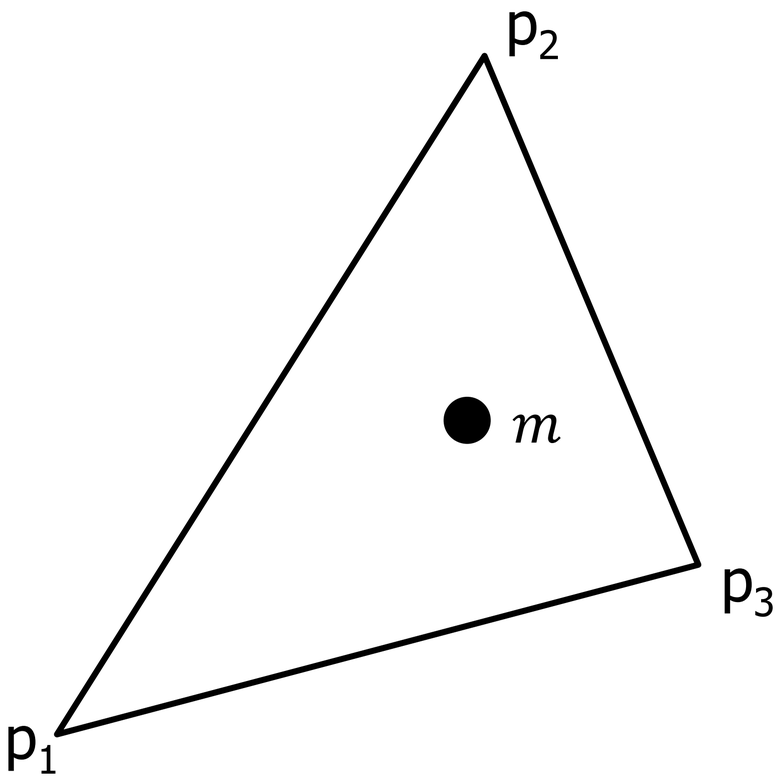
Barycentric Coordinates

- Question: Given a triangle defined by three points p_1, p_2, p_3, how do we associate an inside point with a texture color?



Barycentric Coordinates

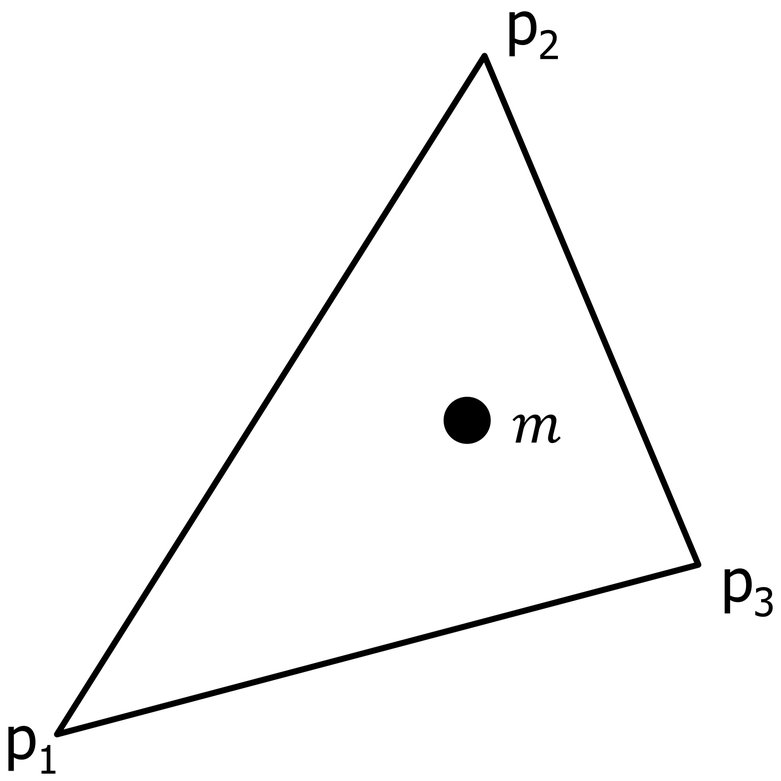
- Solution: Represent the inside point as a weighted sum of the three triangle points. This is called **Barycentric Coordinates**.



$$m = \alpha p_1 + \beta p_2 + \gamma p_3$$

Barycentric Coordinates

- To find the values of (α, β, γ), a set of linear equations needs to be solved.



$$m_x = \alpha x_1 + \beta x_2 + \gamma x_3$$

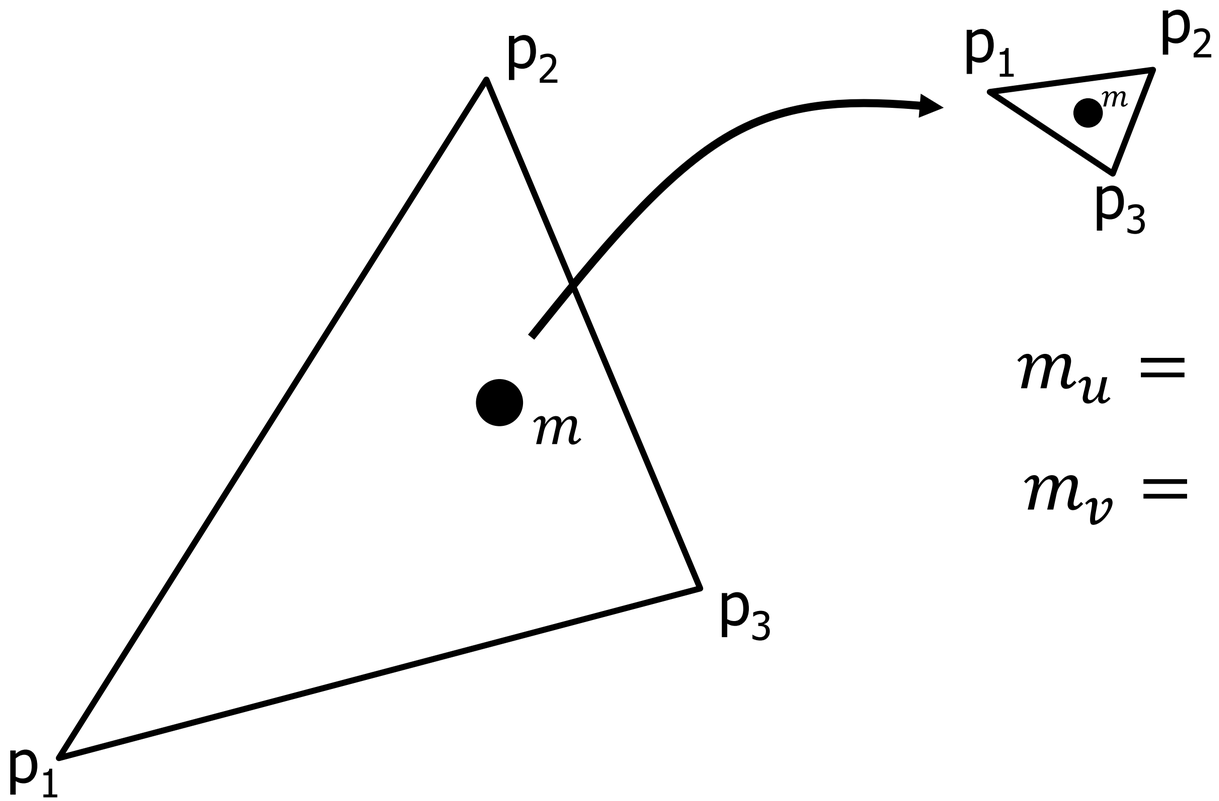
$$m_y = \alpha y_1 + \beta y_2 + \gamma y_3$$

$$1 = \alpha + \beta + \gamma$$

We'll talk more about how to solve this set of equations later…

Barycentric Coordinates

- The point location in uv-space can be found by applying the same weights with uv-coordinates.

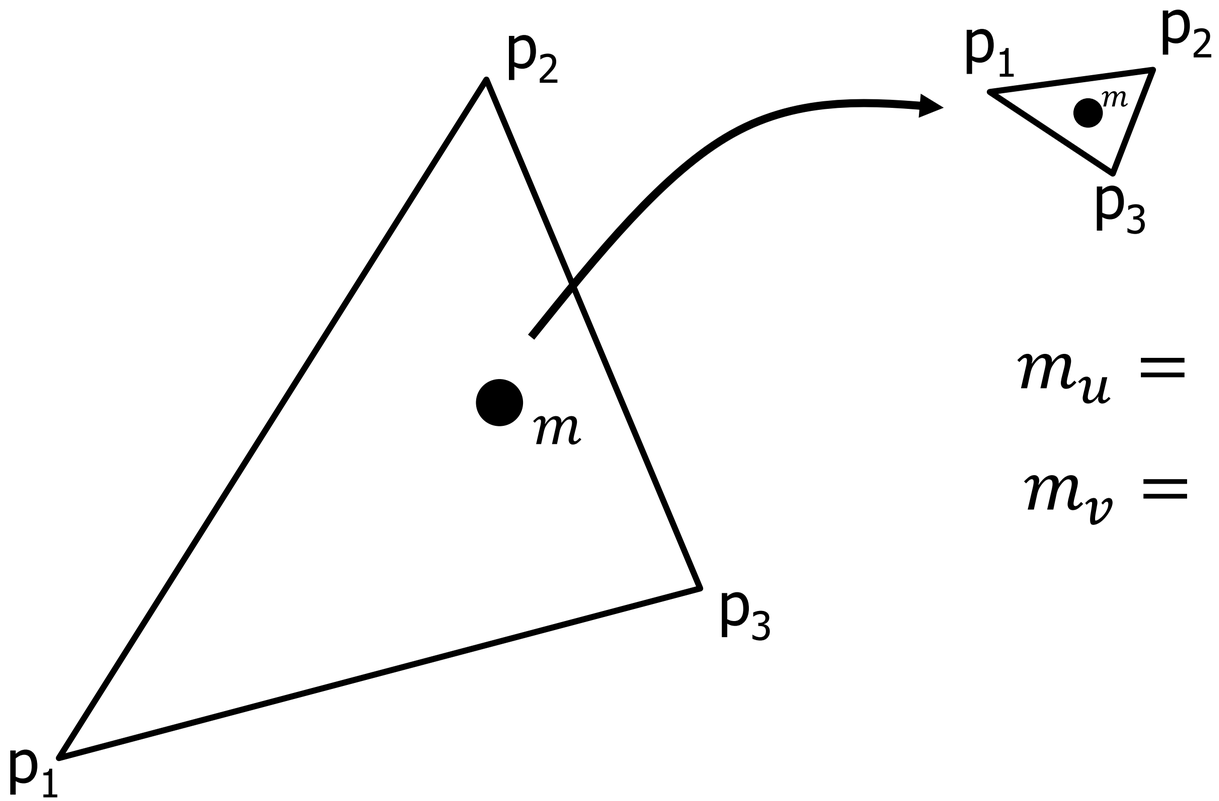


$$m_u = \alpha u_1 + \beta u_2 + \gamma u_3$$

$$m_v = \alpha v_1 + \beta v_2 + \gamma v_3$$

Example

Assume m is determined to be 40% of p_1, 28% of p_2, and 32%, of p_3.



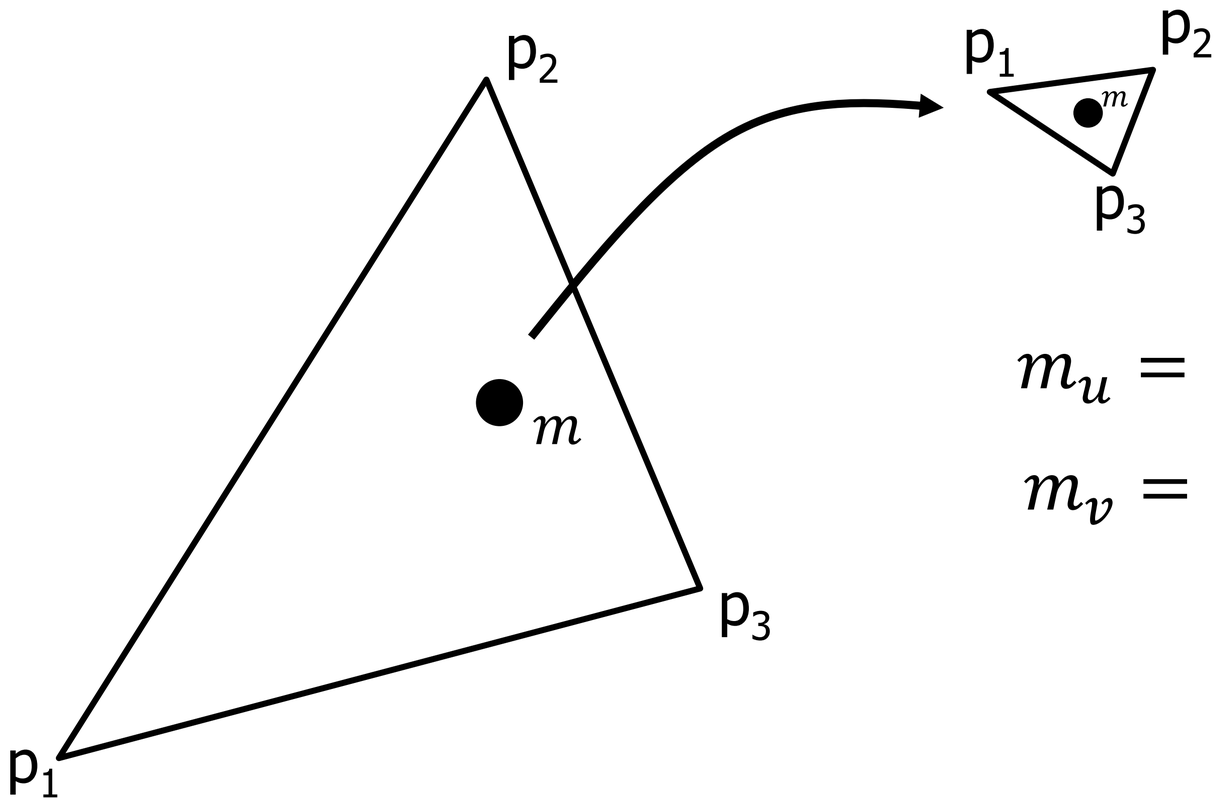
$$m_u = \alpha u_1 + \beta u_2 + \gamma u_3$$

$$m_v = \alpha v_1 + \beta v_2 + \gamma v_3$$

Example

Assume m is determined to be 40% of p_1, 28% of p_2, and 32%, of p_3.

$\alpha = .40, \beta = .28, \gamma = .32$



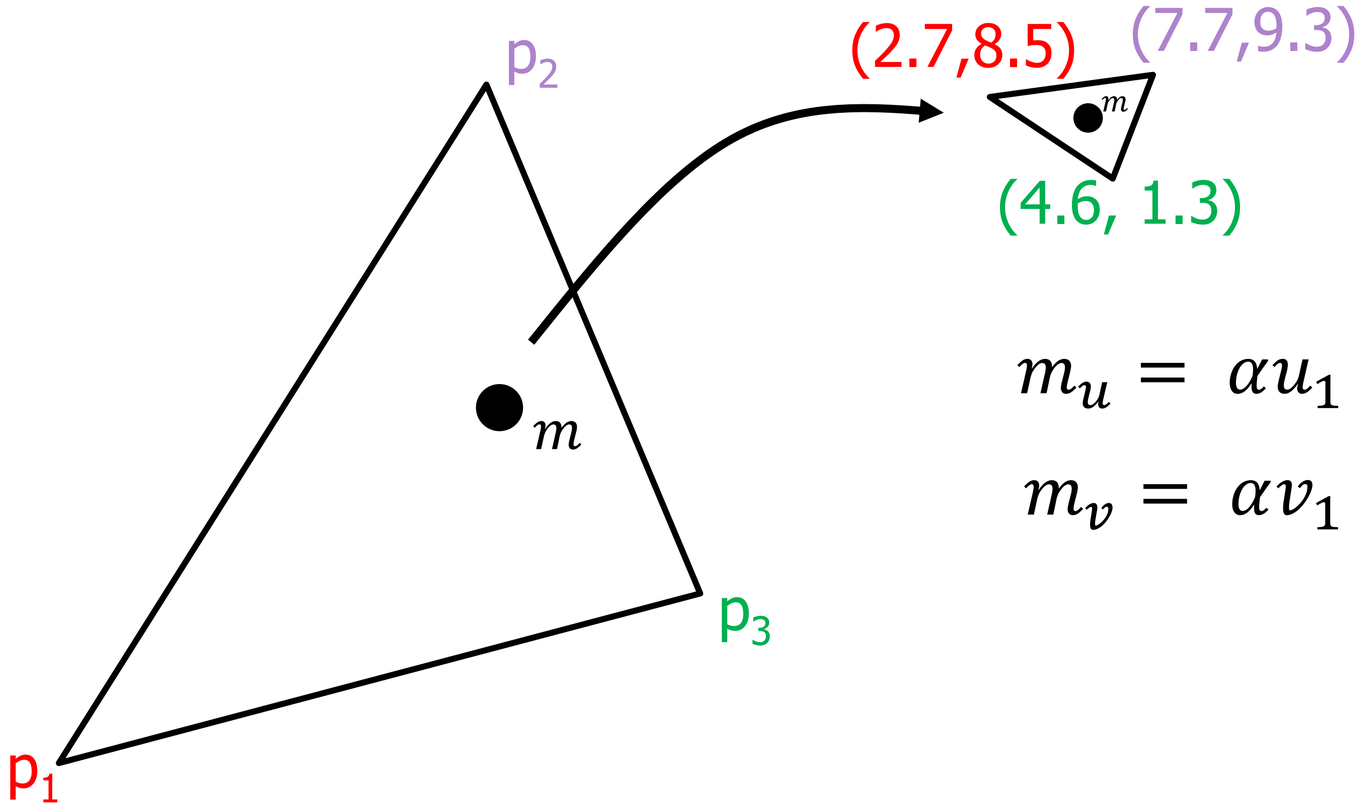
$$m_u = \alpha u_1 + \beta u_2 + \gamma u_3$$

$$m_v = \alpha v_1 + \beta v_2 + \gamma v_3$$

Example

Assume m is determined to be 40% of p_1, 28% of p_2, and 32%, of p_3.

$$\alpha = .40, \beta = .28, \gamma = .32$$



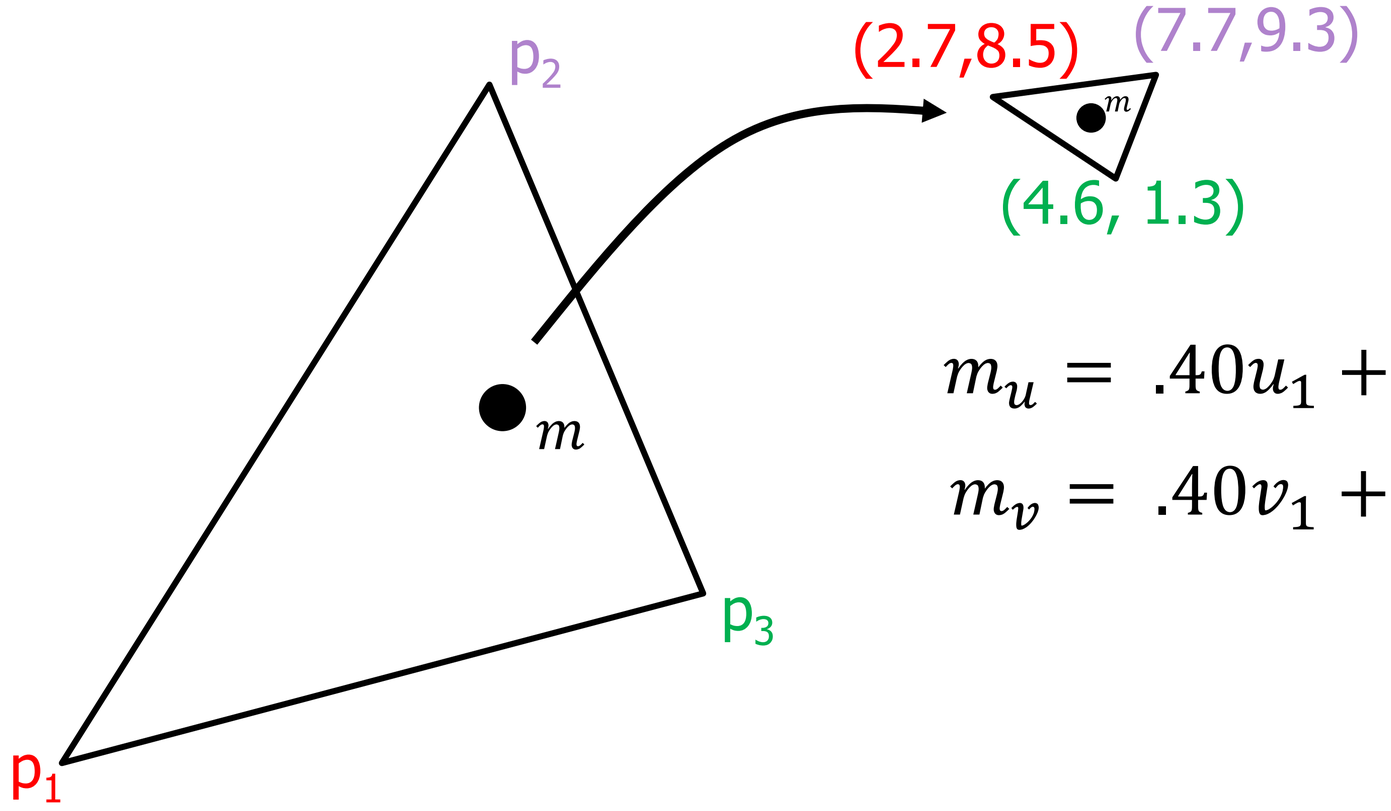
$$m_u = \alpha u_1 + \beta u_2 + \gamma u_3$$

$$m_v = \alpha v_1 + \beta v_2 + \gamma v_3$$

Example

Assume m is determined to be 40% of p_1, 28% of p_2, and 32%, of p_3.

$$\alpha = .40, \beta = .28, \gamma = .32$$



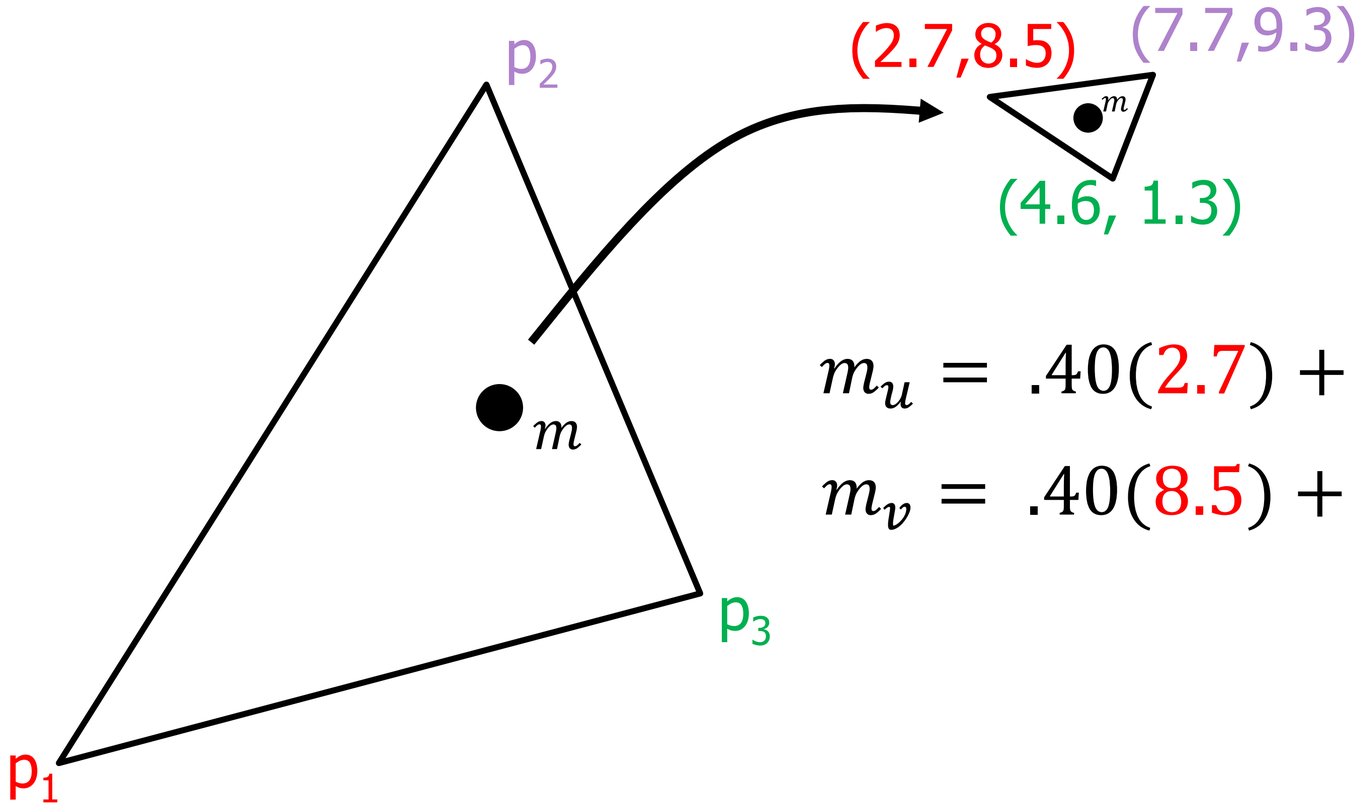
$$m_u = .40u_1 + .28u_2 + .32u_3$$

$$m_v = .40v_1 + .28v_2 + .32v_3$$

Example

Assume m is determined to be 40% of p_1, 28% of p_2, and 32%, of p_3.

$$\alpha = .40, \beta = .28, \gamma = .32$$



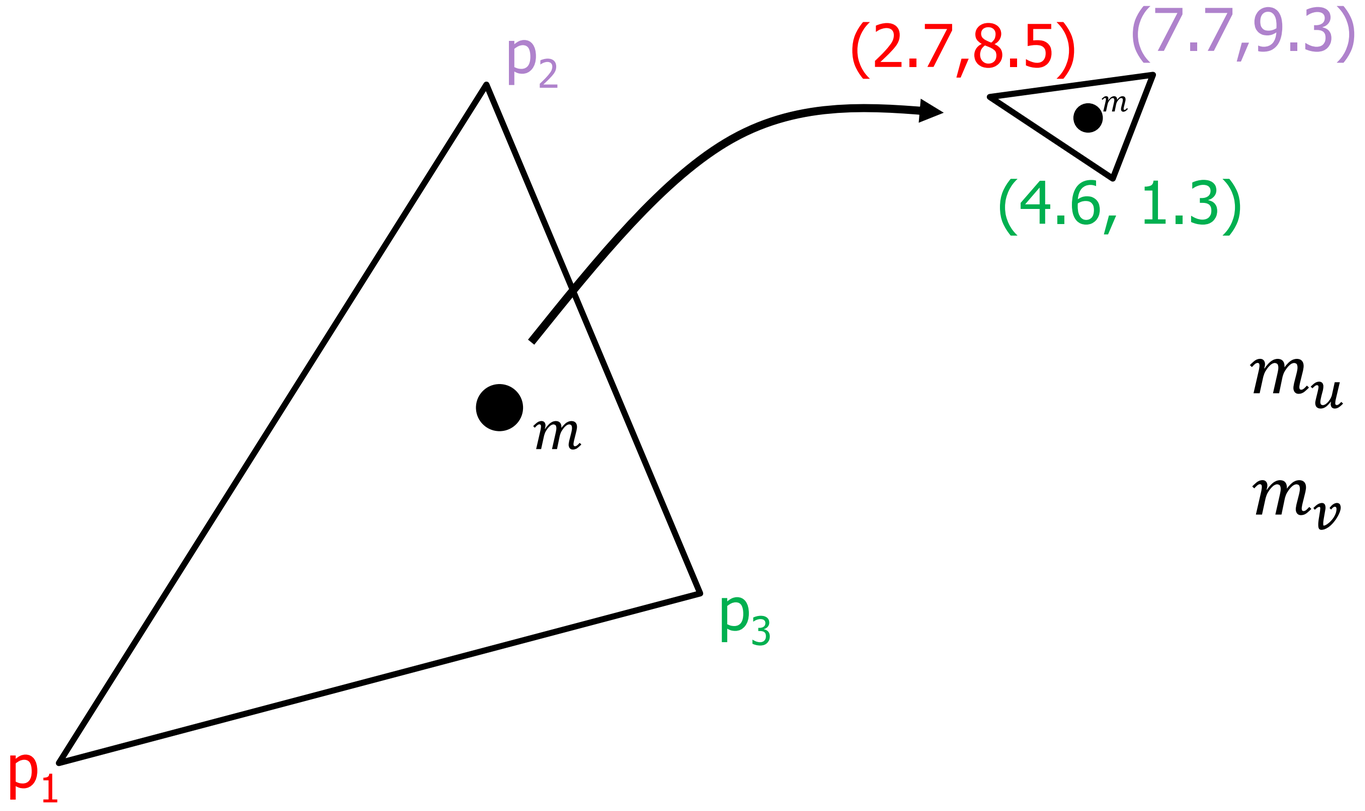
$$m_u = .40(2.7) + .28(7.7) + .32(4.6)$$

$$m_v = .40(8.5) + .28(9.3) + .32(1.3)$$

Example

Assume m is determined to be 40% of p_1, 28% of p_2, and 32%, of p_3.

$$\alpha = .40, \beta = .28, \gamma = .32$$



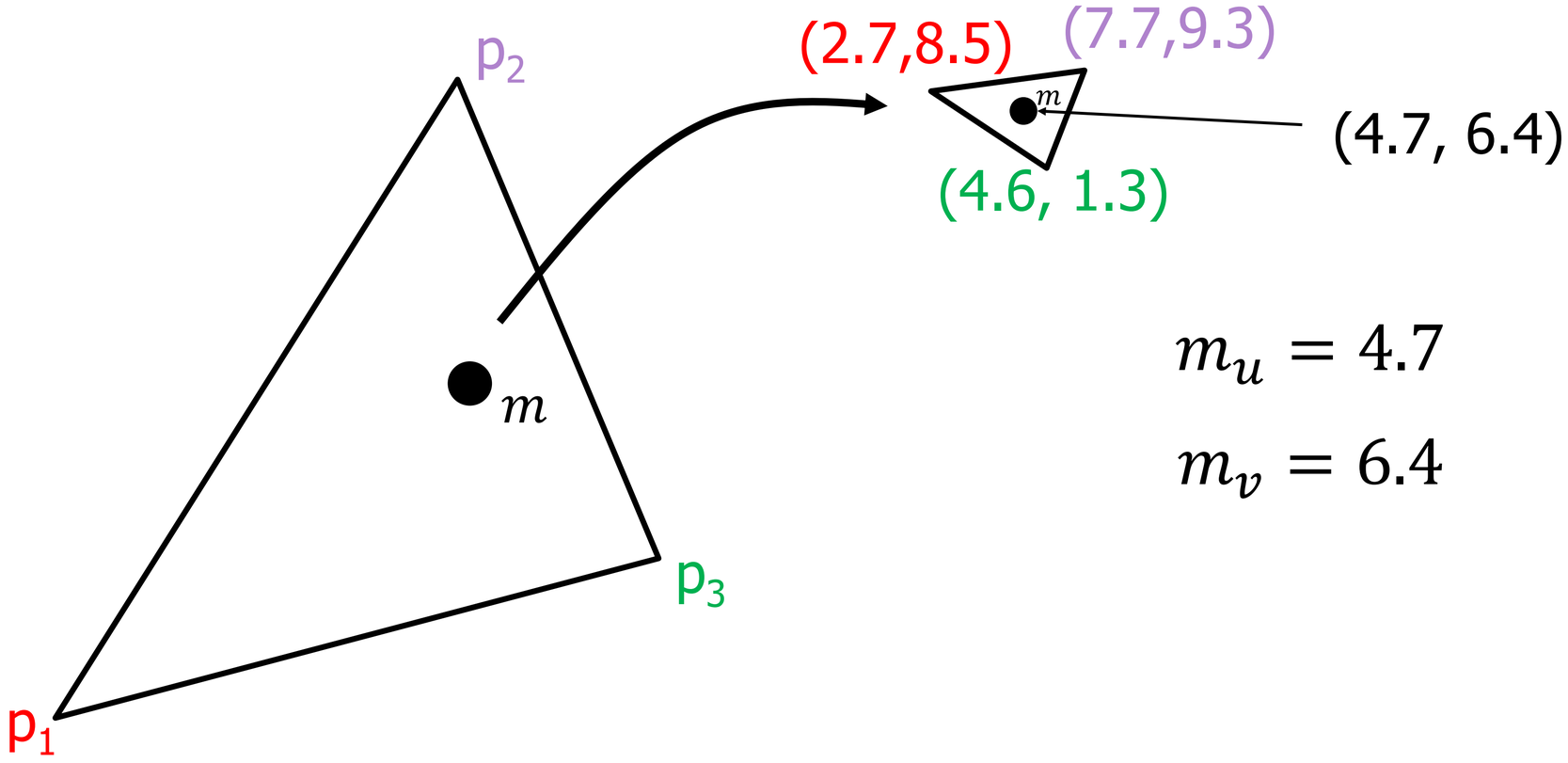
$$m_u = 4.7$$

$$m_v = 6.4$$

Example

Assume m is determined to be 40% of p_1, 28% of p_2, and 32%, of p_3.

$$\alpha = .40, \beta = .28, \gamma = .32$$



$$m_u = 4.7$$

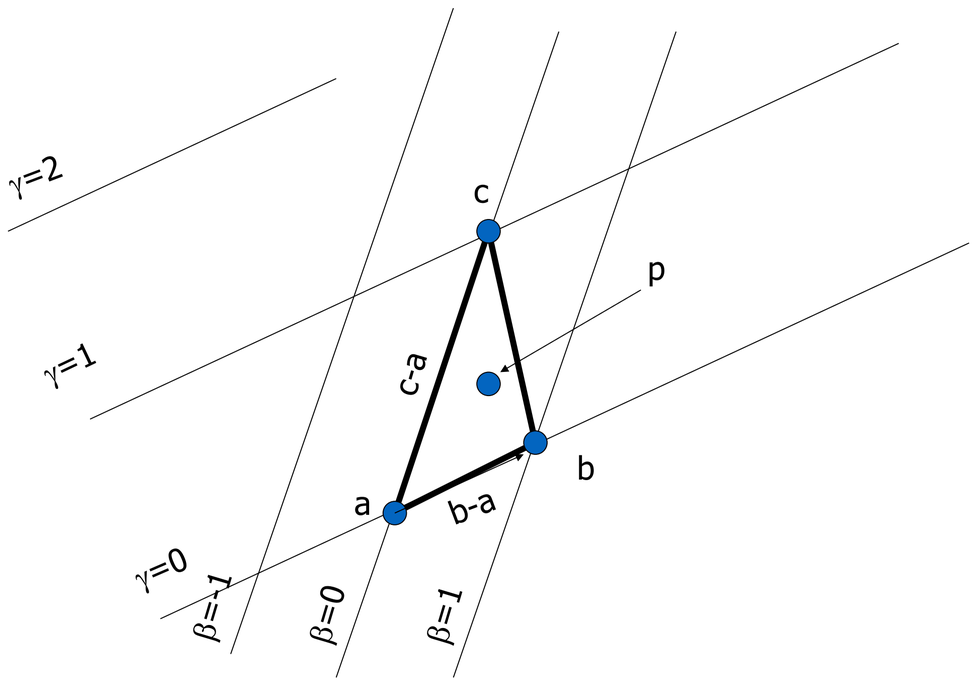
$$m_v = 6.4$$

Barycentric: Alternate Formulation

$$p = a + \beta(b-a) + \gamma(c-a)$$

- Think of one vertex of the triangle as the origin, and two sides of your triangle as a set of non-orthogonal axes.

- The inside point is defined as a (β, γ) offset from the origin.



Texture Painting

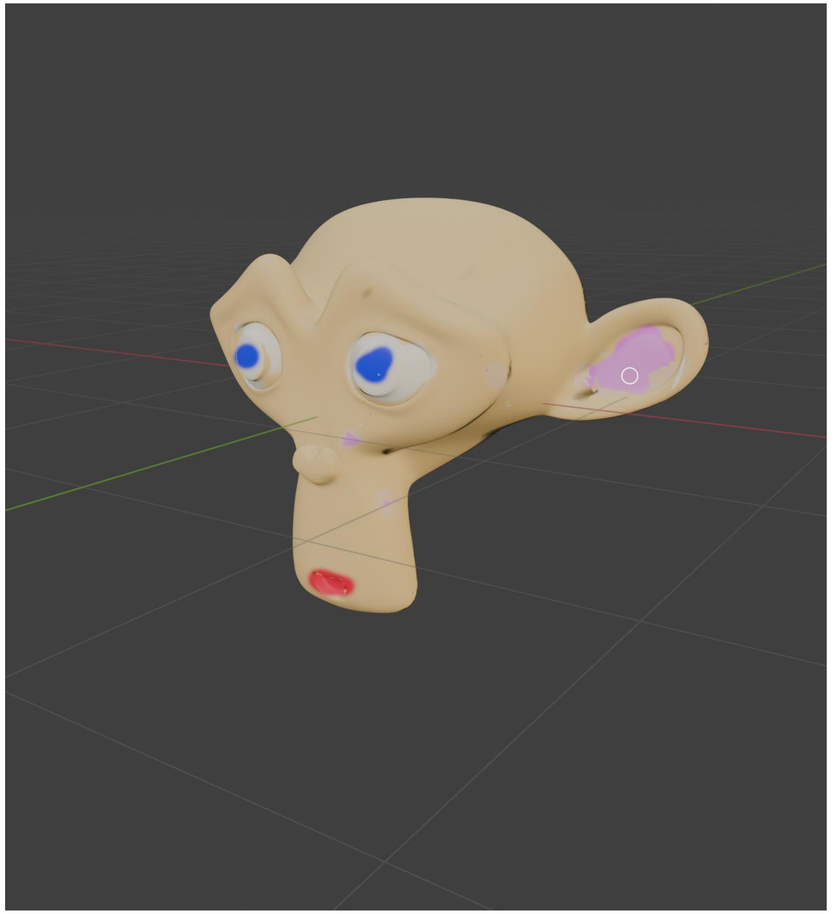
CSCI 3800/6820: Computer Graphics

Outline

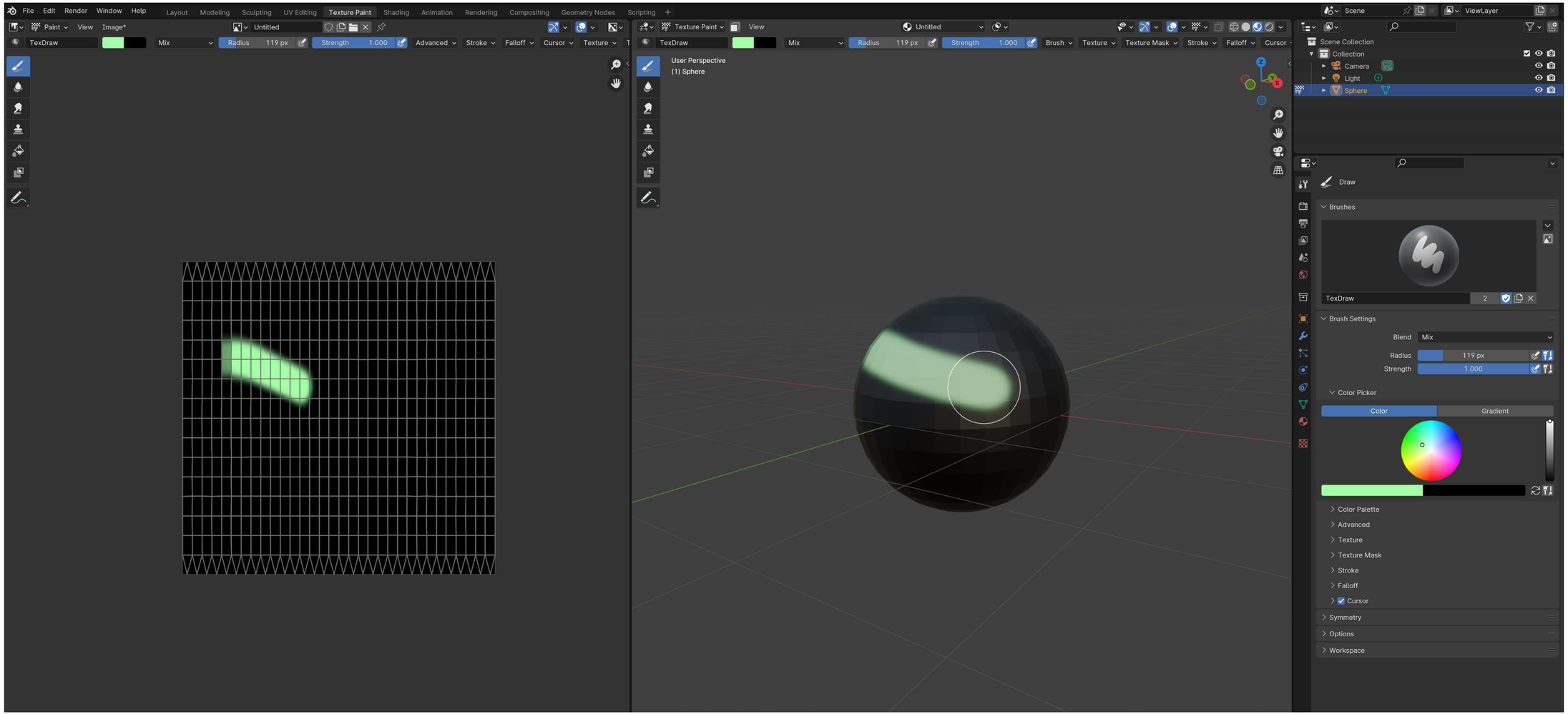
- Texture Painting
- Materials
  - Roughness
  - Index of Refraction (IOR)
  - Bump and Displacement Mapping
- Kinds of Texture Maps

Texture Painting

- Texture Painting allows you to color 3D models with more intuitive controls.
- Specifically, it lets you color in a 3D space.
- It is designed to feel like you are airbrushing or spray painting the 3D model.



The Interface



How do you get effects like this?

Material Properties

- Color is not the only detail that matters in describing a surface.
- Light can change the perceived color of an object based on properties such as refraction and reflectance.
- These can be defined as part of the texture of the object.

https://blenderdiplom.com/en/downloads/584-download-cycles-material-test-scene.html

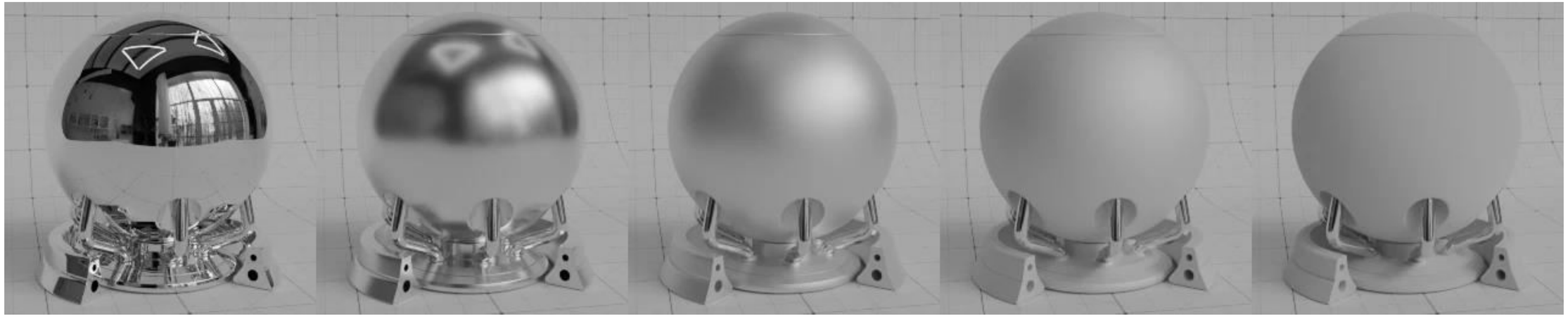
Material Properties

- Material properties include
  - Base Color
  - Roughness
  - Index of Refraction (IOR)
  - And much more…

https://blenderdiplom.com/en/downloads/584-download-cycles-material-test-scene.html

Roughness

- How much the surface directly reflects light.
- 0.0 gives a perfect mirror.

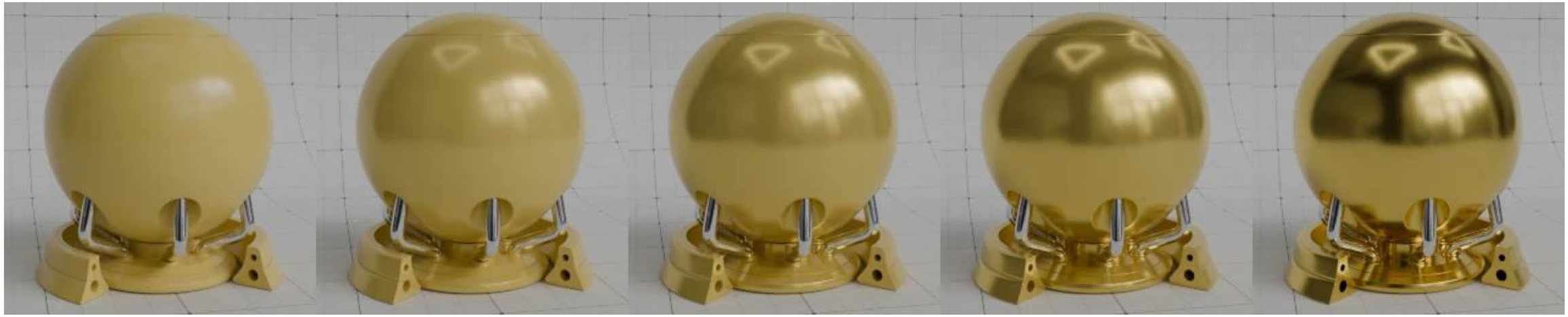


0.0 → **1.0**

https://docs.blender.org/manual/en/latest/render/shader_nodes/shader/principled.html

Metallic

- How much the surface reflections tint in the base color.
- Based on physical models of metals.

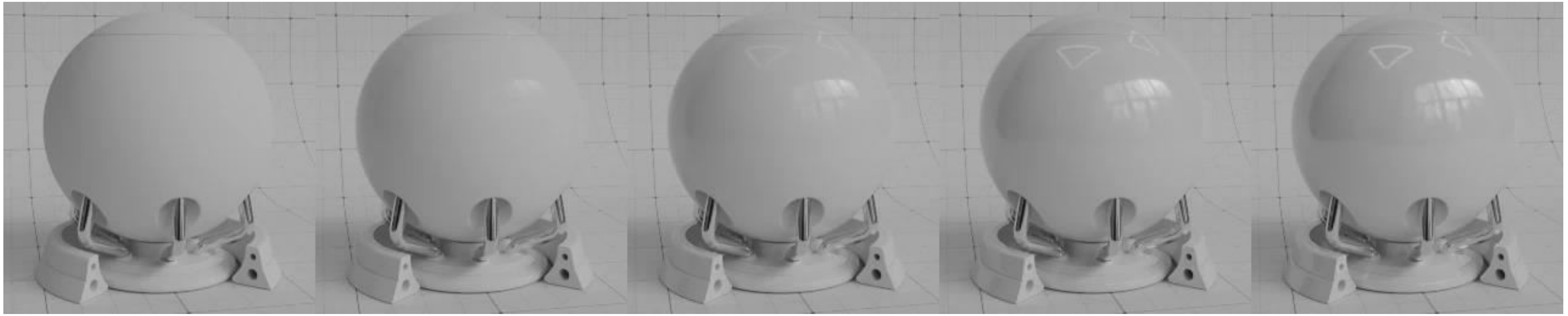


0.0 → **1.0**

https://docs.blender.org/manual/en/latest/render/shader_nodes/shader/principled.html

Index of Refraction (IOR)

- How much light is redirected within an object.
- Based on physical models of glass.

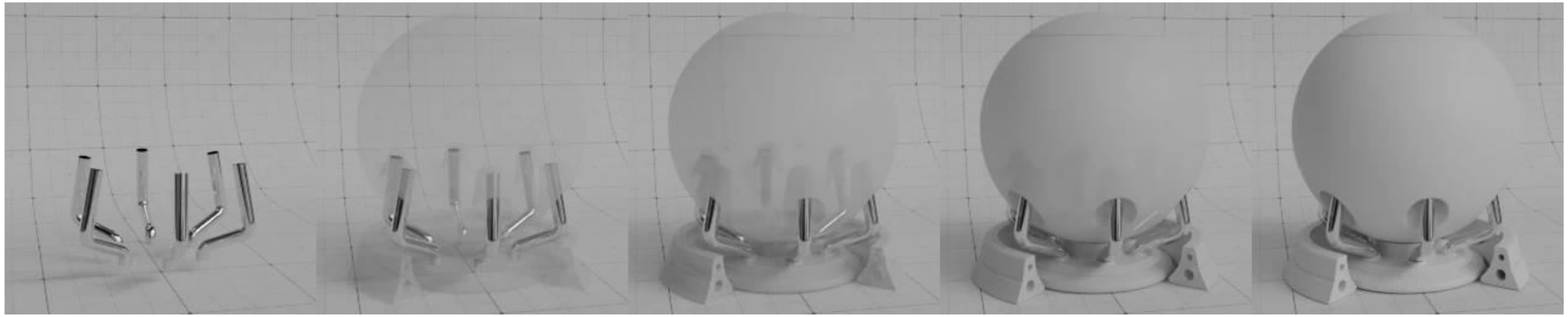


1.0 → **2.0**

https://docs.blender.org/manual/en/latest/render/shader_nodes/shader/principled.html

Alpha

- How transparent or opaque an object is.
- Generally only used with texture maps or specific styles.



0.0 → **1.0**

https://docs.blender.org/manual/en/latest/render/shader_nodes/shader/principled.html

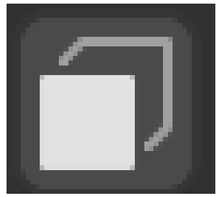
Demo: Hammer

- Texture Painting
- Masked Painting
- Material Properties

Credit: Grant Abbitt

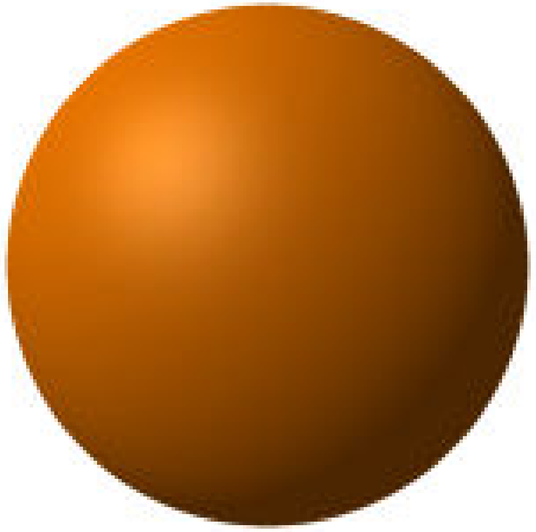
Quick Tips

- Make sure you create a material, add a new base color image, and unwrap on the UV tab before you texture paint.
- Turn on or off masked painting by clicking on the icon on the top tool bar.



- Set the strength to 1.0 to start, then turn it back down for more subtle highlights.
- When selecting faces for masked painting, go to Wireframe View and Edit Mode to make those selections.

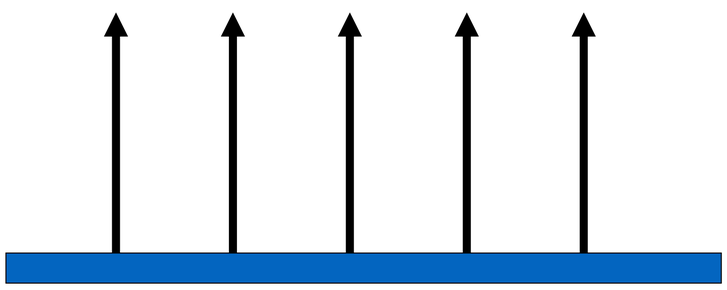
Bump Mapping



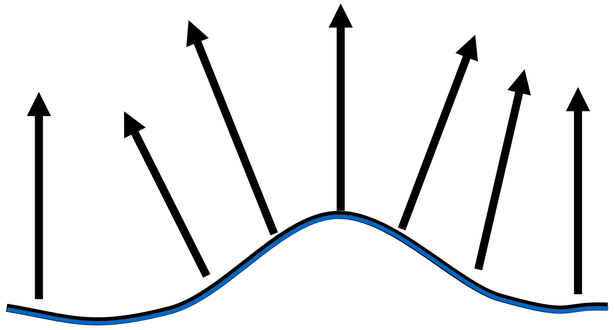
- Texture mapping adds color detail to the object, but the object still looks *smooth.*
- How do you make the object look “rough”?
- Solution 1: Add more polygons
  - Too time intensive
- Solution 2: Bump mapping

Bump Mapping

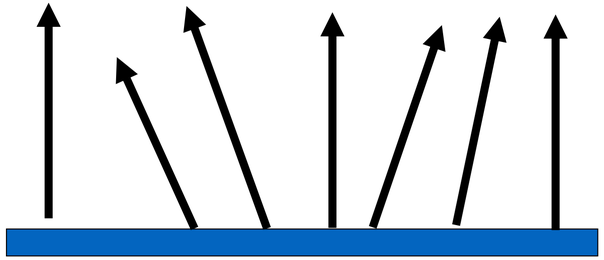
- **Idea**: If we use If we use “fake” normals, we can trick the renderer into thinking there are bumps on the surface.



Flat Plane

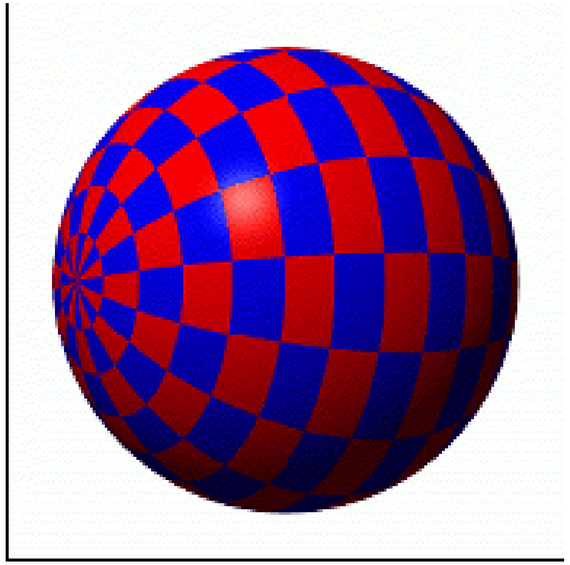


Real Bump

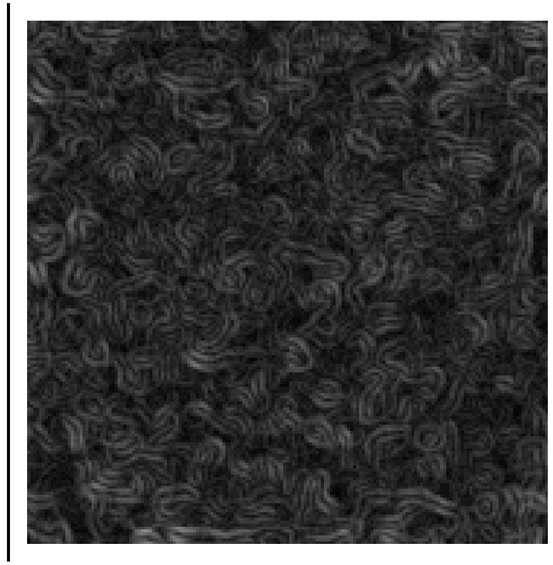


Fake Bump

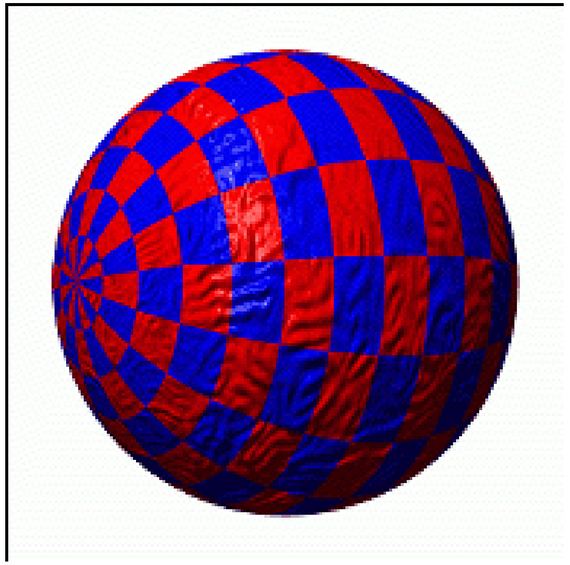
Bump Mapping Examples



Sphere with Texture Map

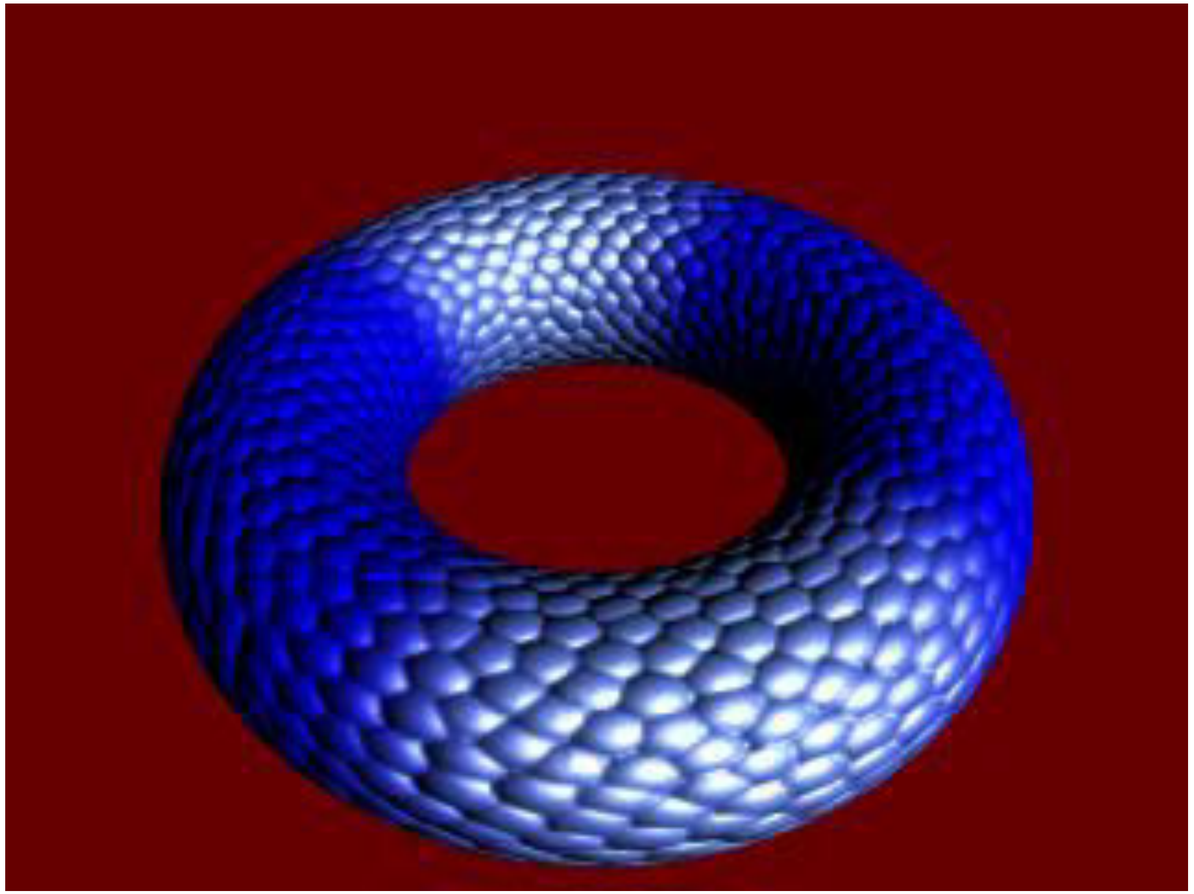


Bump Map

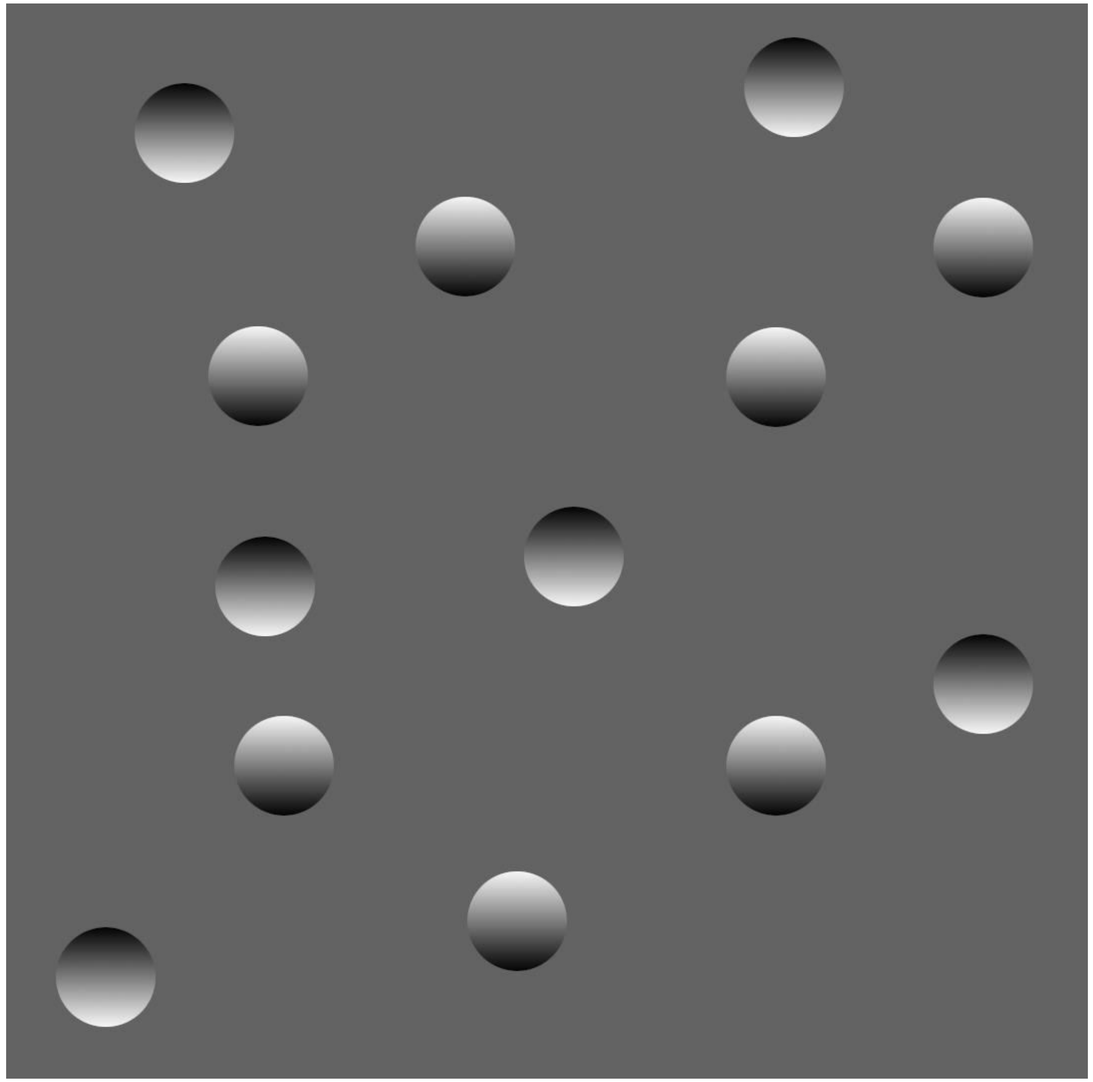


Sphere with Texture and Bump Maps

Bump Mapping Examples

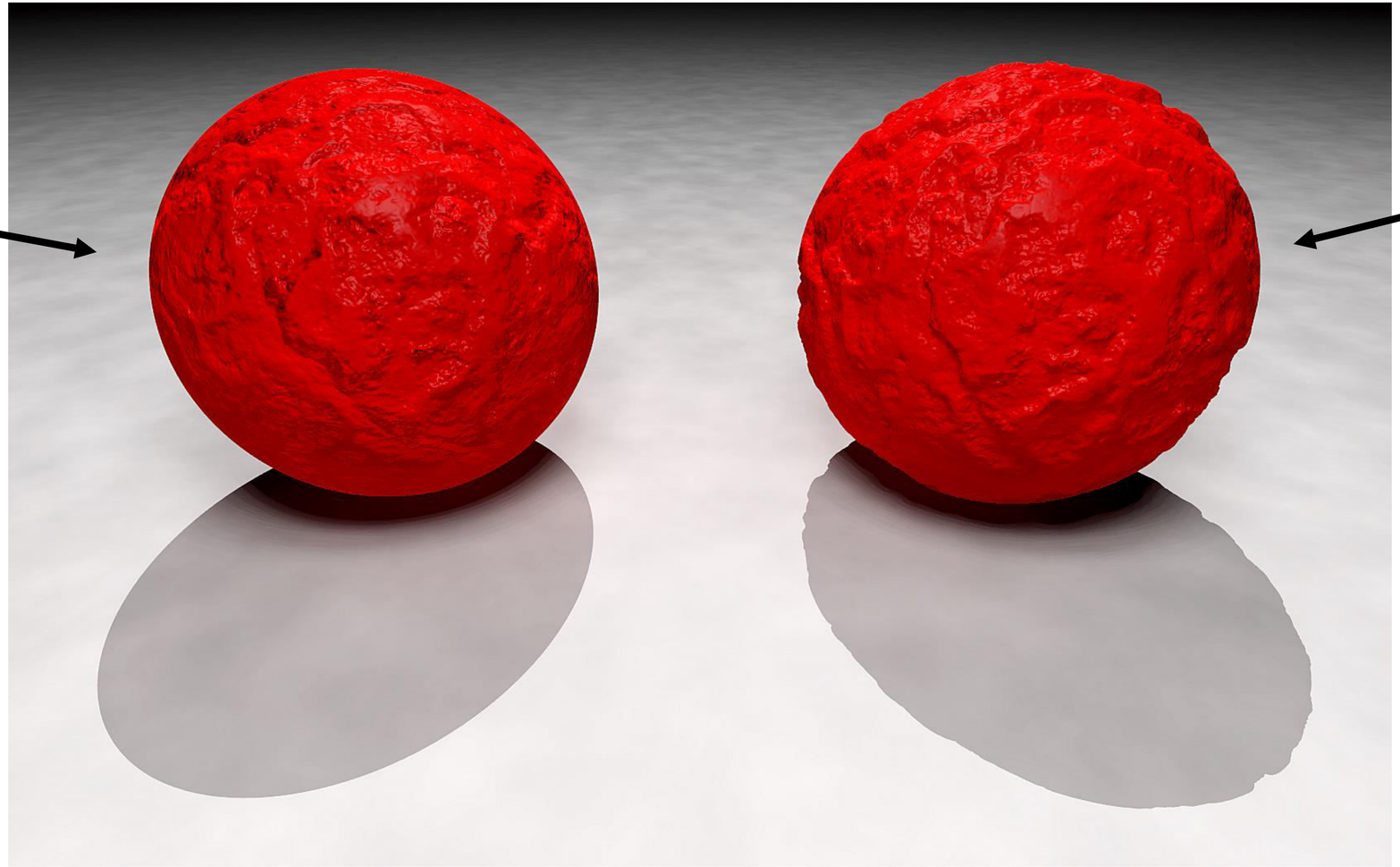


It’s an optical illusion…



Warning: Silhouettes

Bump Map

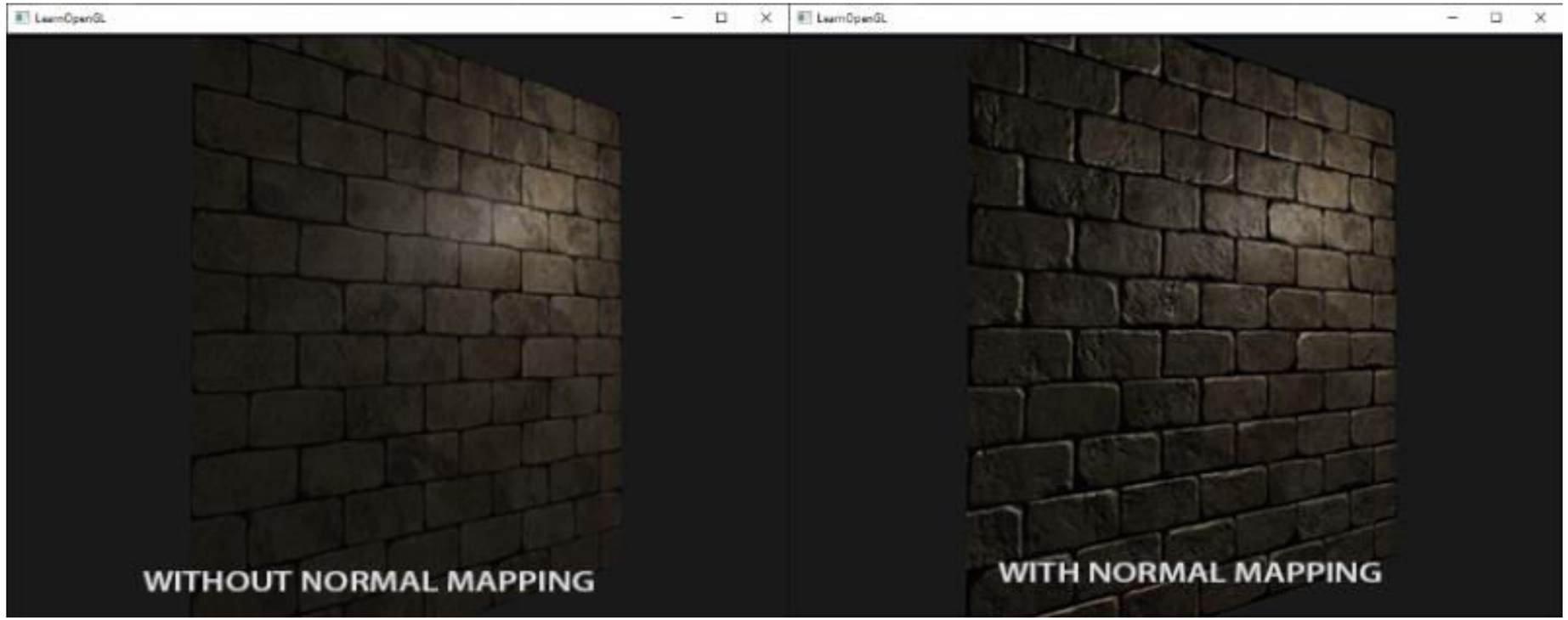


Actual Geometry

Bump maps don’t change geometry, so they can’t affect shadows.

Normal Maps

- Like bump mapping, but instead of describing an amount of “bump” at each location, an x,y,z vector is given describing the normal to the surface.

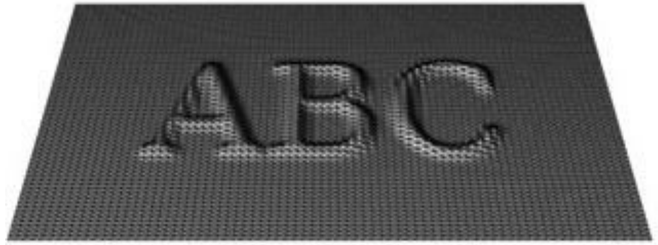


Displacement Mapping

- The actual geometry is altered in order to allow for a more “bumpy” look
- A “displacement map” is given, along with the surface
- The displacement map is applied to the surface, and additional geometry is introduced where needed
- Gives a more realistic look, particularly at the silhouette
- Quite a bit more costly than the other techniques

ORIGINAL MESH

DISPLACEMENT MAP



MESH WITH DISPLACEMENT

Comparison

http://www.chromesphere.com/Tutorials/Vue6/Optics_Basics_Print.html

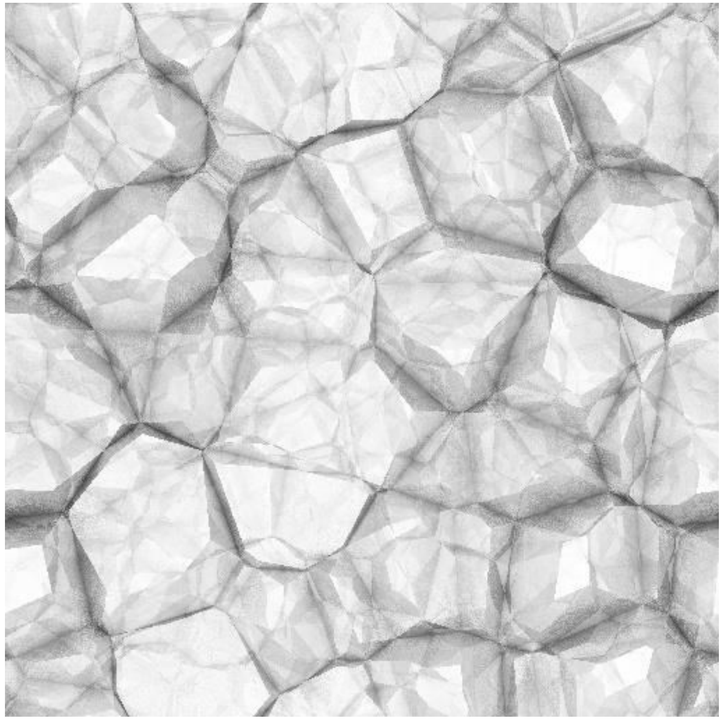
Other Mapping Techniques

- Other mapping techniques exist for modifying the rendered surface, but normal maps and displacement maps are the most common.

Bump Mapping | Horizon Mapping | Displacement Mapping | View Dependent Displacement Mapping

Ambient Occlusion (AO) Map

- Determines how much the scene's ambient light affects the color.
- Used to give the illusion of self-shadowing.
- Sometimes called "baking in" soft shadows.



https://www.filterforge.com/filters/10500-ambientocclusion.html

Summary of Texture Maps

- Ambient Occlusion/Roughness/Metal (arm) – A texture that combines three properties that determine how light interacts with the surface.
- Diffuse (diff) – The base color of the surface
- Normal (norm) – A kind of bump mapping that redirects light as it hits the surface.
- Displacement (disp) – If present, physically modifies the geometry at render time.

Demo: Loading Texture Maps

Loading Texture Maps: Option 1

- Download .blend file from website such as polyhaven.com
- In blender, click File -> Append
- Click on .blend file, and navigate to Material
- Click on the object of interest, and select the imported material in the materials tab.

Loading Texture Maps: Option 2

- Enable “Node Wrangler” under Edit -> Preferences
- Download .blend file from website such as polyhaven.com
- Click on the object of interest, then go to the Shading tab.
- Make new material in the bottom of the screen.
- Click on the “Principled BSDF” tab, and Ctrl + Shift + T
- Select all texture images you wish to import

Note: Displacement Maps

- Displacement maps can be tricky to configure in Blender. To help make displacement maps display properly, try the following
  - Switch the renderer from EVEE to Cylces
  - Switch the viewport display modes to “Rendered”
  - In the Materials menu under Displacement, swith “Bump Only” to “Displacement Only”
  - Use 3D rectangle objects instead of planes for floor and walls.

Bump Mapping: The Math

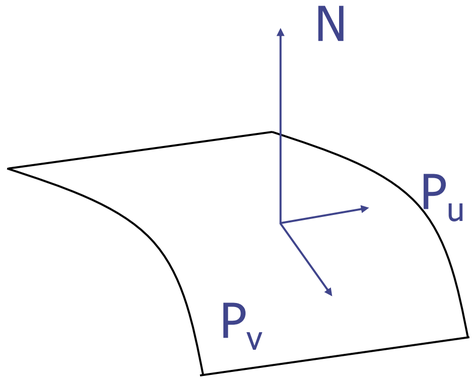
- If you want to dive deeper into the math of Bump Mapping, follow these slides by Professor Parris Egbert of Brigham Young University.

The Bump Map

- The bump map is defined as an array of displacements *B(u,v)*
- If *P(u,v)* specifies positions on a parametric surface, the normal at the surface can be calculated as

$$\vec{N} = \vec{P}_u \times \vec{P}_v$$

- where $\vec{P}_u$ and $\vec{P}_v$ are partial derivatives of the surface at point *P* lying in the tangent plane



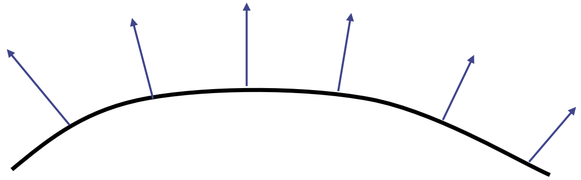
Bump Mapping

- We can simulate bumps by adding small perturbations to the surface positions along the direction of the normal

$$P'(u,v) = P(u,v) + B(u,v)\frac{N}{|N|}$$

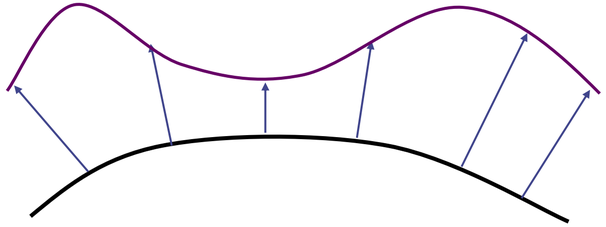
Bump Mapping

- So far we have:



Original surface P

Bump map B



Bump map applied
to surface giving P’

Bump Mapping

- Next we compute new normals for bump mapped surface

$$\vec{N}' = \vec{P}'_u \times \vec{P}'_v$$

- What are the partials?

$$P_u'(u,v) = \frac{\partial}{\partial u}\left[P(u,v) + B(u,v)\frac{N}{|N|}\right]$$

$$= P_u(u,v) + B_u(u,v)\frac{N}{|N|} + B(u,v)\frac{N_u}{|N|}$$

$$P_v'(u,v) = \frac{\partial}{\partial v}\left[P(u,v) + B(u,v)\frac{N}{|N|}\right]$$

$$= P_v(u,v) + B_v(u,v)\frac{N}{|N|} + B(u,v)\frac{N_v}{|N|}$$

- To simplify, we approximate, using as the reason that the bump should be significantly smaller than the surface. Thus, we remove the last terms of the equations, giving:

$$P_u'(u,v) \approx P_u(u,v) + B_u(u,v)\frac{N}{|N|}$$

$$P_v'(u,v) \approx P_v(u,v) + B_v(u,v)\frac{N}{|N|}$$

- Now, since

$$\vec{N}' = \vec{P}'_u \times \vec{P}'_v$$

$$N' \approx \left(P_u + B_u \frac{N}{|N|} \right) \times \left(P_v + B_v \frac{N}{|N|} \right)$$

$$= P_u \times P_v + B_v \left(P_u \times \frac{N}{|N|} \right) + B_u \left(\frac{N}{|N|} \times P_v \right) + B_u B_v \left(\frac{N}{|N|} \times \frac{N}{|N|} \right)$$

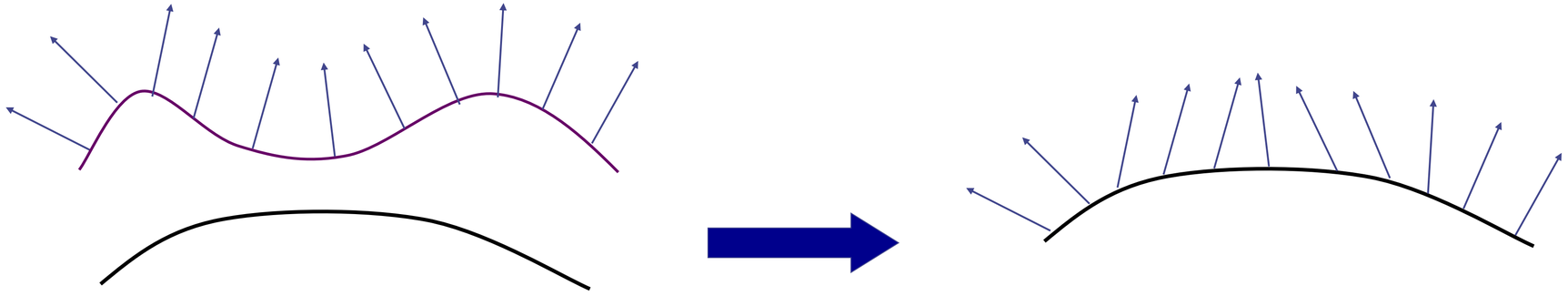
- And, since

$$\frac{\vec{N}'}{\left|\vec{N}\right|} \times \frac{\vec{N}'}{\left|\vec{N}\right|} = \vec{0}$$

then

$$N' \approx N + B_v\left(P_u \times \frac{N}{|N|}\right) + B_u\left(\frac{N}{|N|} \times P_v\right)$$

- This gives:



- This gives a normal perturbation that is:
  - independent of the position of the object
  - independent of the orientation of the object
- i.e., a position on the object will always get the same perturbation

Lighting and Shading

CSCI 3800/6820: Computer Graphics

Outline

- Shading Objects
  - Ambient vs Diffuse vs Specular Coloring
- Types of Lighting
  - Point, Sun, Spot, Area
  - Customized Lights (IES)
  - Customized Shapes (Emission)
- Environment Maps (HDRIs)

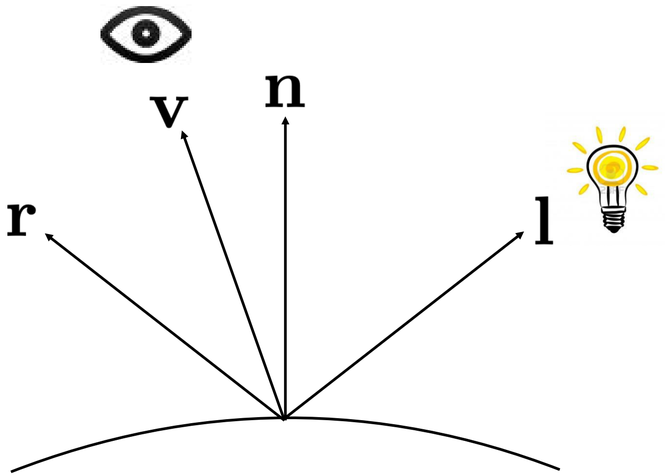
Shading Objects

- The color that a surfaces appears depends on two factors:
  - The material properties of object.
  - The lighting conditions of the scene.
- Determining the final color of an object in a scene is called **shading.**

Examples

Basic Geometry of Lighting

- The surface **n**ormal
- The **l**ighting direction (to the light)
- The **v**iewing direction (to the eye/camera)
- The **r**eflected light direction

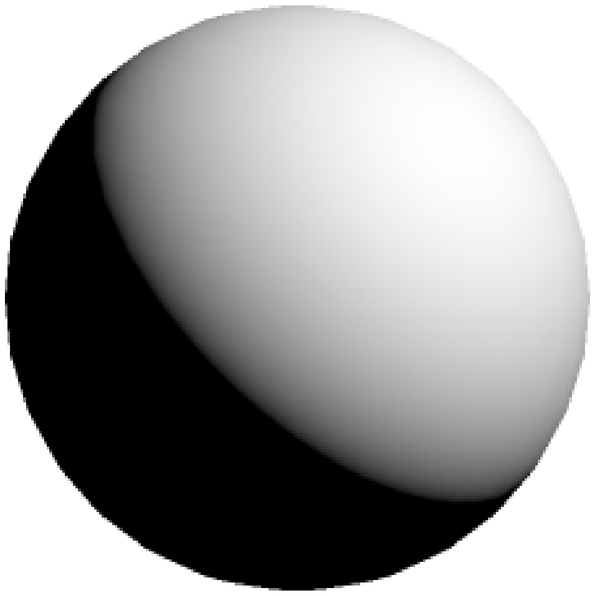


Components of Shading

- In the most basic models, the final appearance color is mixture of three components that are blended:
  - **Ambient** – The pure base color without regard to specific lights in the scene. Usually, it is a small percentage to account for ambient light present in most scenarios.
  - **Diffuse** – The base color based on the amount of light shining on the surface.
  - **Specular** – Specific reflections based on the viewing angle of the camera.

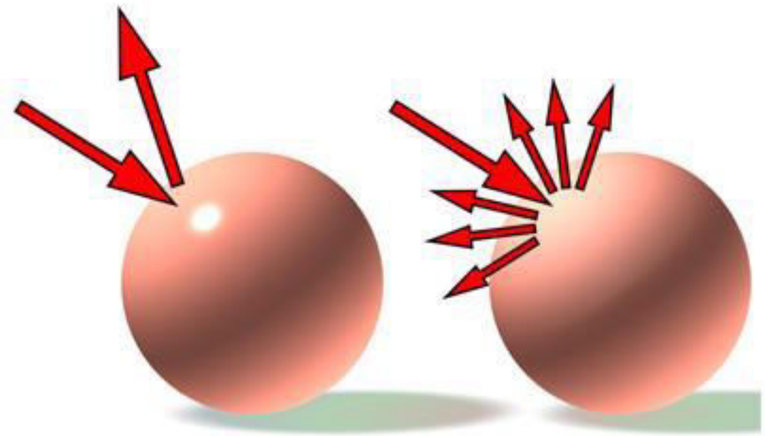
Diffuse Reflection

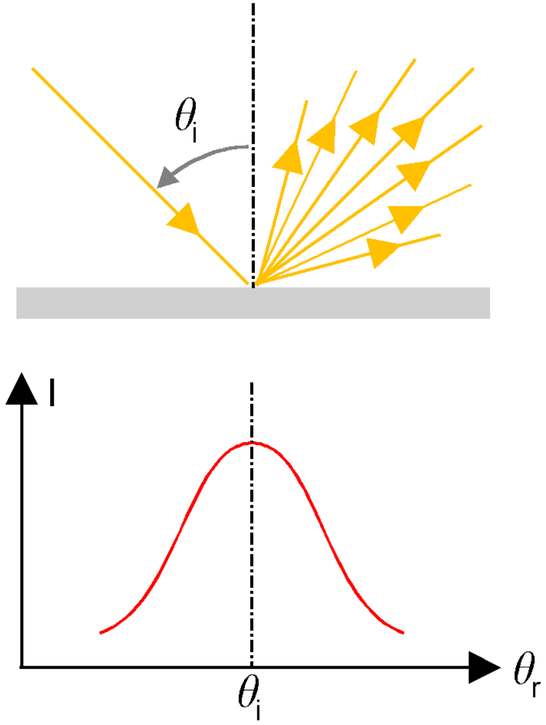
- Light scattered in every direction is called the *diffuse* part of the reflected light
- A perfectly diffuse surface is called *Lambertian*
- Only lighting direction matters
- Viewing direction *does not*



Specular Reflections

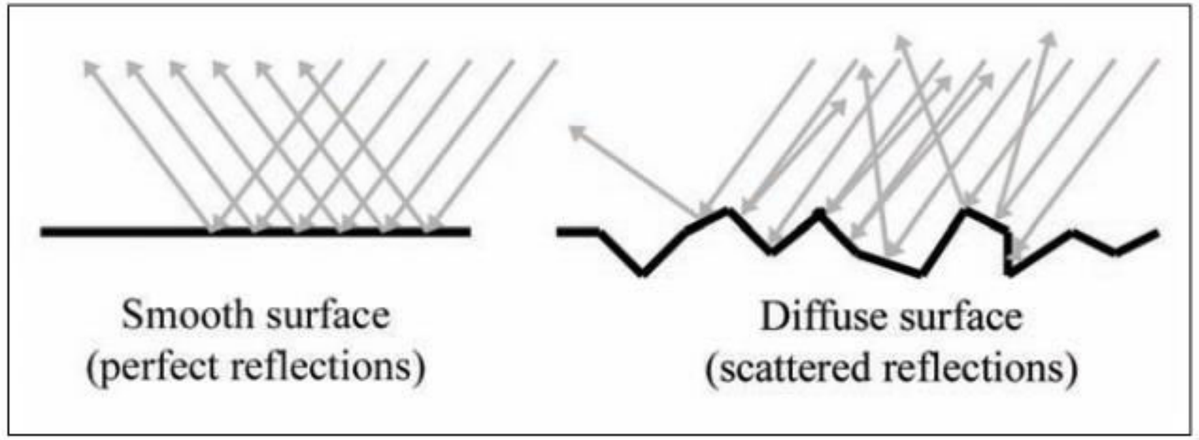
Specular Reflections





Angle of reflection = Angle of incidence
(but may be blurred)

Specular vs. Diffuse

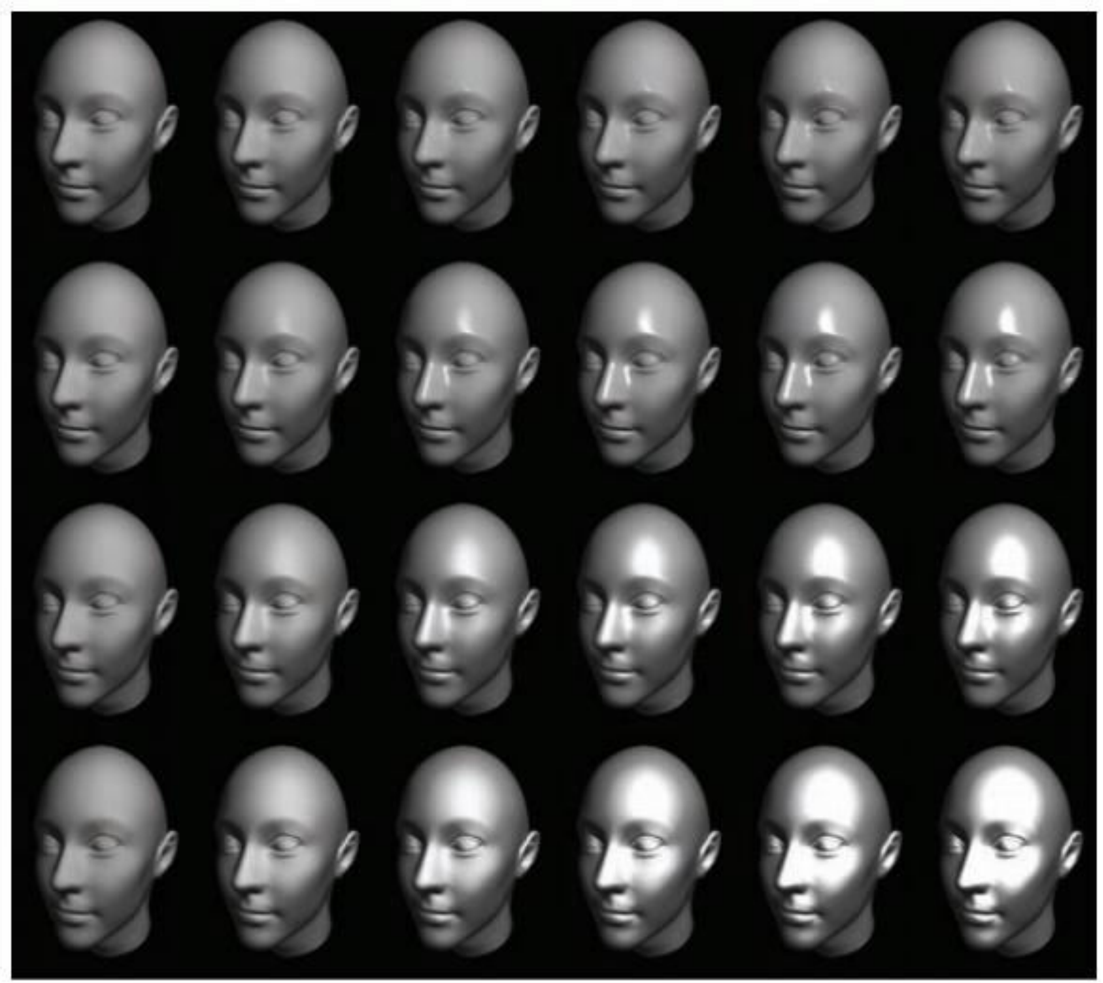


Some light is reflected perfectly (specular)

Some light is scattered (diffuse)

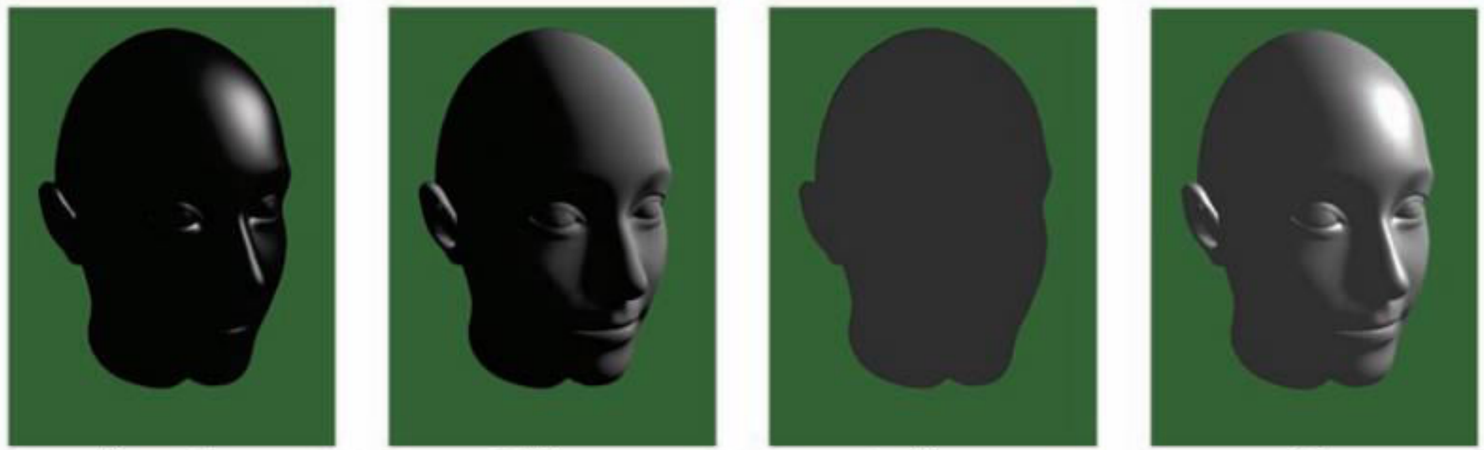
Specular Reflections

m_{gls}



$\mathbf{m}_{\mathrm{spec}}$

All Together Now…



This is called the *Phong* model
(the *Blinn* model is similar with slightly different specular)

BRDFs

- The Phong model is only an approximation
- Real reflections are not a simple mix of pure diffuse and pure specular
- Function of both incoming direction and outgoing direction
- Reflectance isn’t constant across the surface

BRDFs

- The goal is to determine the output color given the
  - Location of the surface
  - The direction of all incoming lights
  - The intensity and wavelengths of all incoming lights
  - The direction to the viewing camera
- The properties of the material help to define this function.

position on surface

wavelength

$$f(\mathbf{x}, \omega_1, \omega_2, \lambda)$$

in out

Bidirectional Reflectance Distribution Function

Kinds of Lighting

- Direct:
  Light falling on an object directly from a light source

- Indirect:
  Light falling on an object after being reflected off
  (or going through) other objects

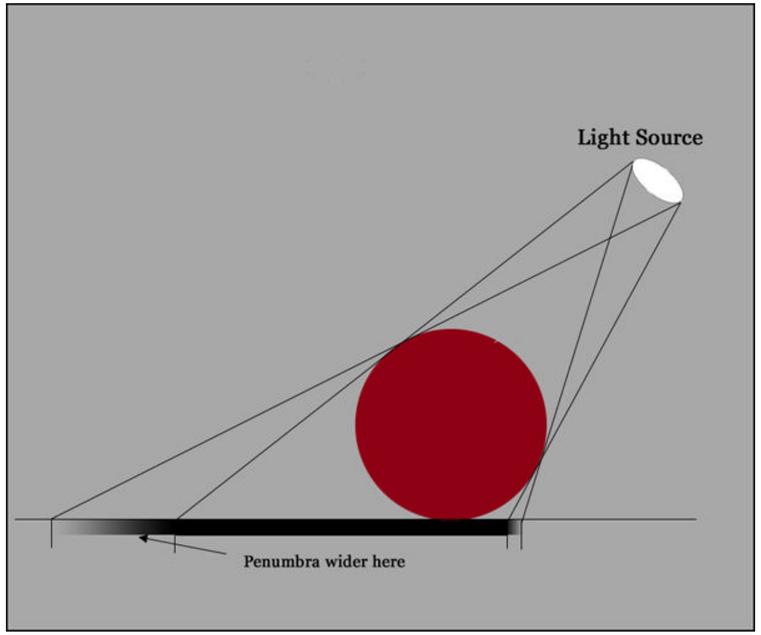
- Ambient:
  General light bouncing around and scattered enough to be effectively
  “everywhere”

Light Sources

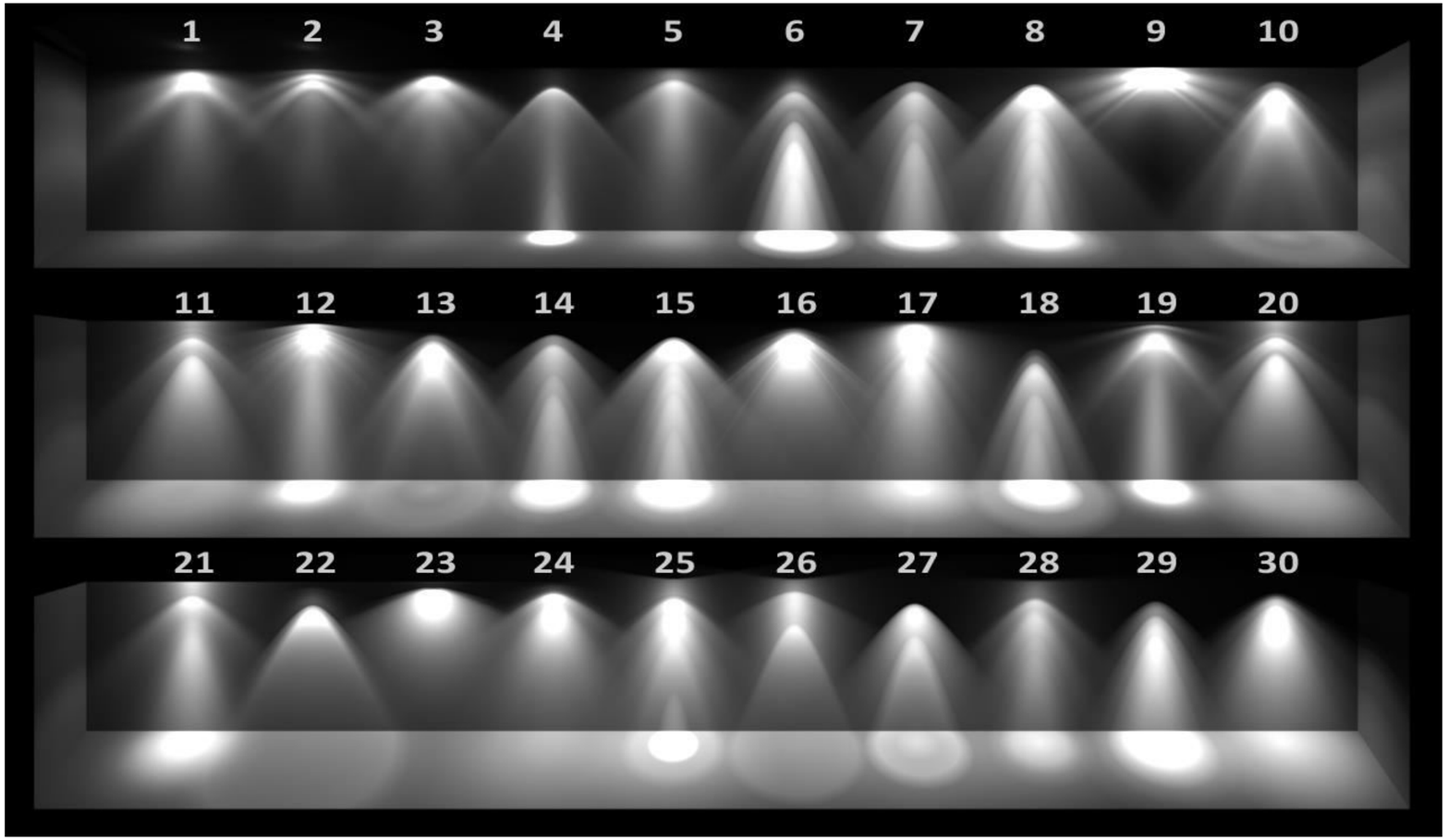
- Point – Light emits radially from a single location
- Sun/Directional – Light emits only in a single direction
- Area – Light emits radially from all points on a surface.
- Spot – Light emits at specific angles from a single location.
- and many other models...

What About Shadows?

- Point lights cast hard shadows
- Area lights cast softer shadows
  - Umbra = area in full shadow
  - Penumbra = area in partial shadow



IES Files



https://help.autodesk.com/view/ARNOL/ENU/?guid=arnold_core_ac_photometric_light_html

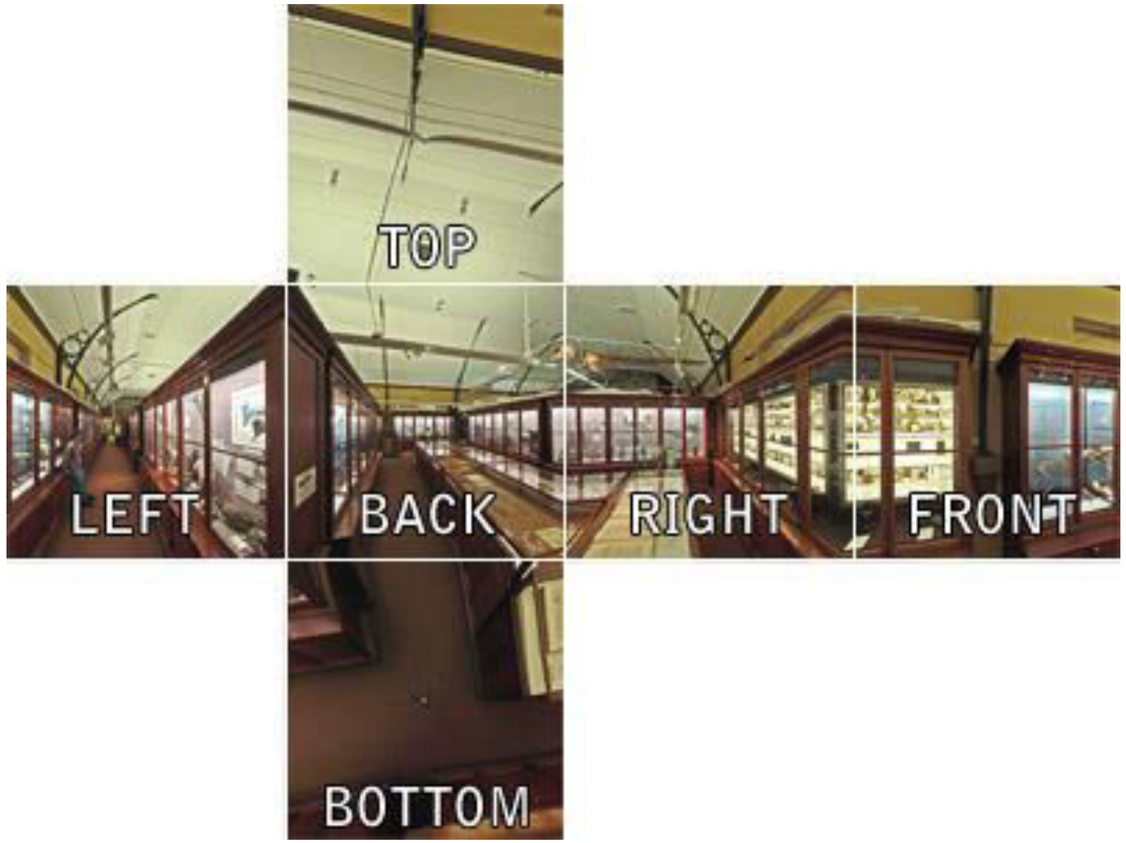
- Physical lights have very specific emission properties based on temperature, angle, and wavelength.
- IES files can be used to import such properties to lights in the scene.

Customized Shapes

- Any shape can be turned into a light source in Blender.
- Use the “Emission” setting under the Materials tab.
- Great for making neon signs.

Environment Maps

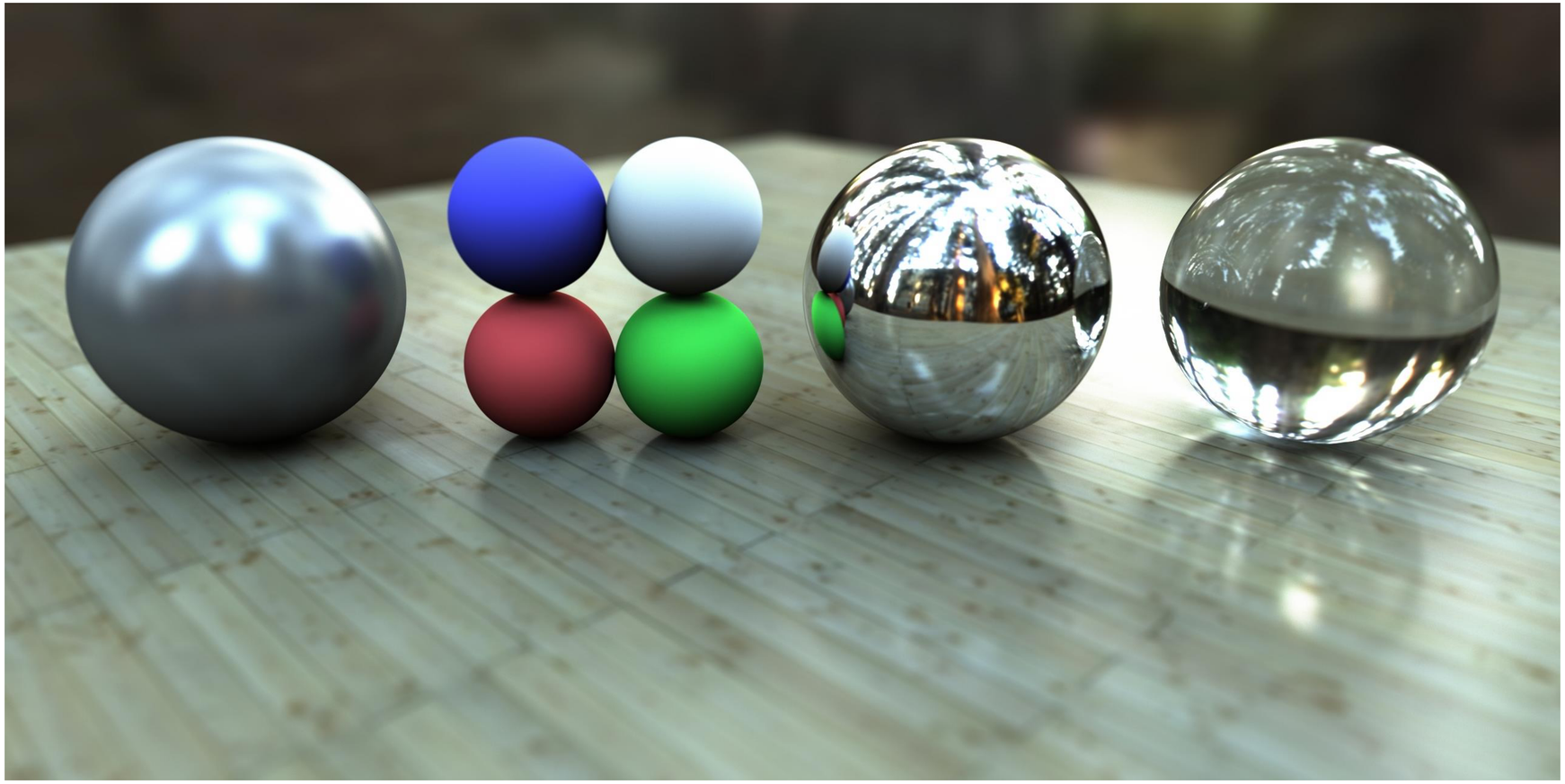
- Environment Maps define the light coming from the boundaries of the scene.
- Can be represented as a cube map or planar (equirectangular) projection.
- Best results occur when the environment map is a **High Dynamic Range Image (HDRI).**



HDRI Environment Map

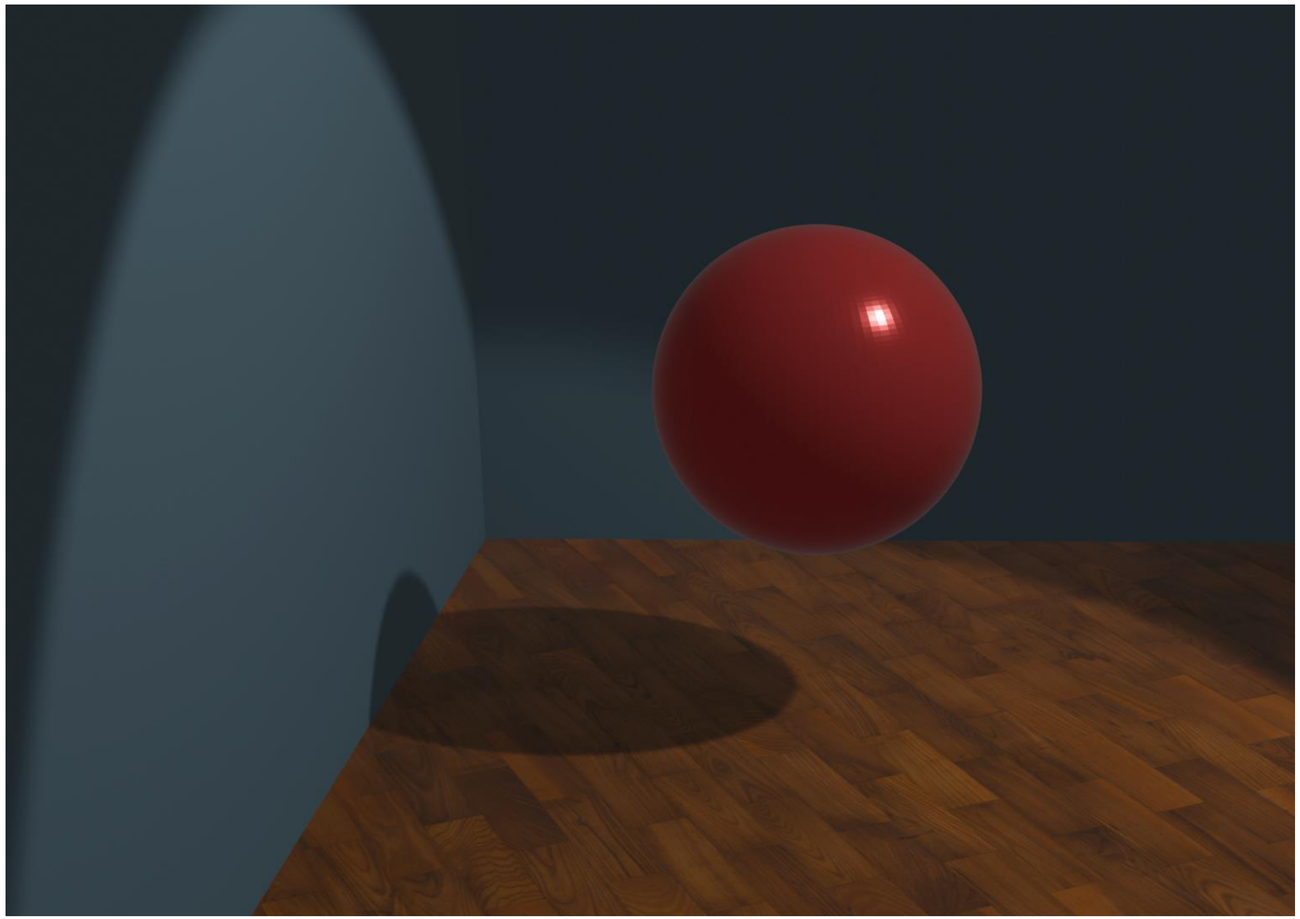
HDRI Environment Map

Environment Maps



Demo: Turn on the Lights

- Adding lights to a scene
- Controlling color and strength
- Importing HDRI images
- Making customized lights



Quick Tips

- The angle of a point light does not matter, only the location does.
- The location of a sun light does not matter, only the angle does.
- Increase the power of your lights to at least 100W.
- Important HDRI files as Environment Textures.

Rendering

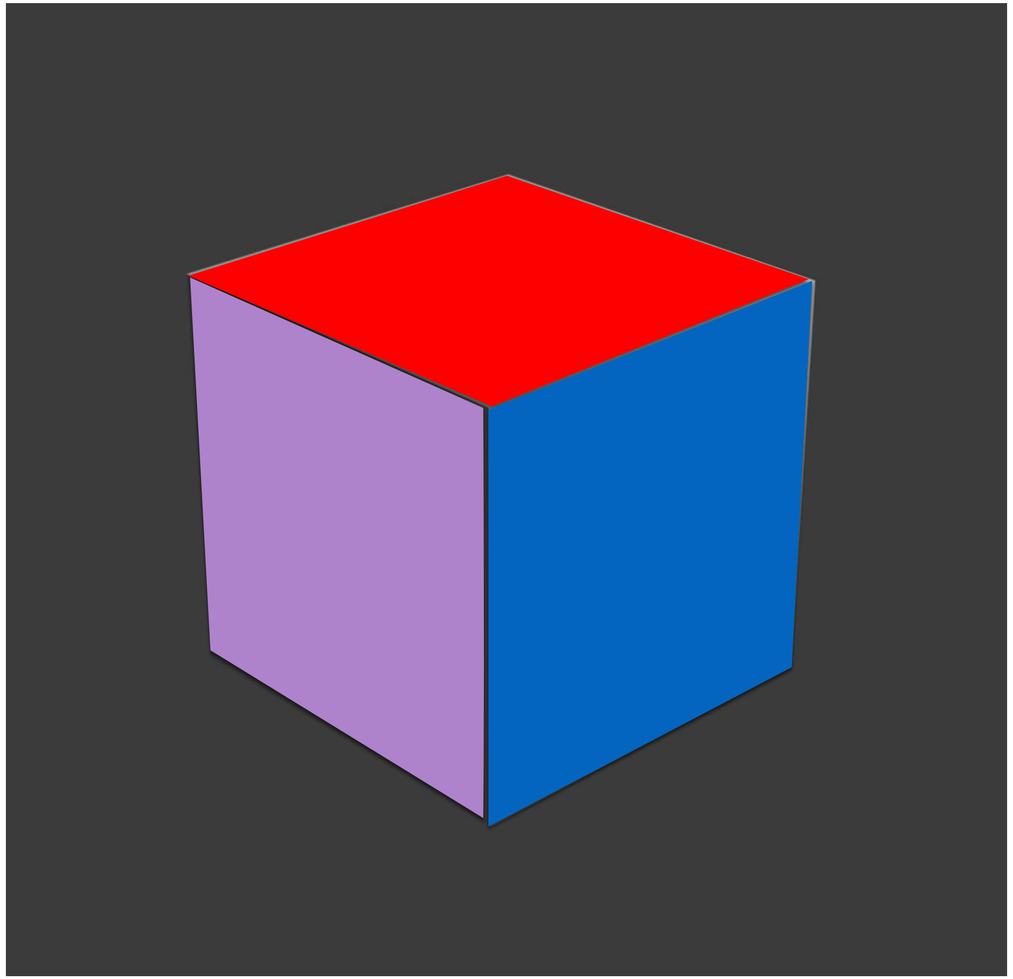
CSCI 3800/6820: Computer Graphics

Outline

- Types of Rendering
  - Raster-based
  - Ray Tracing
- Configuring a Ray Tracer

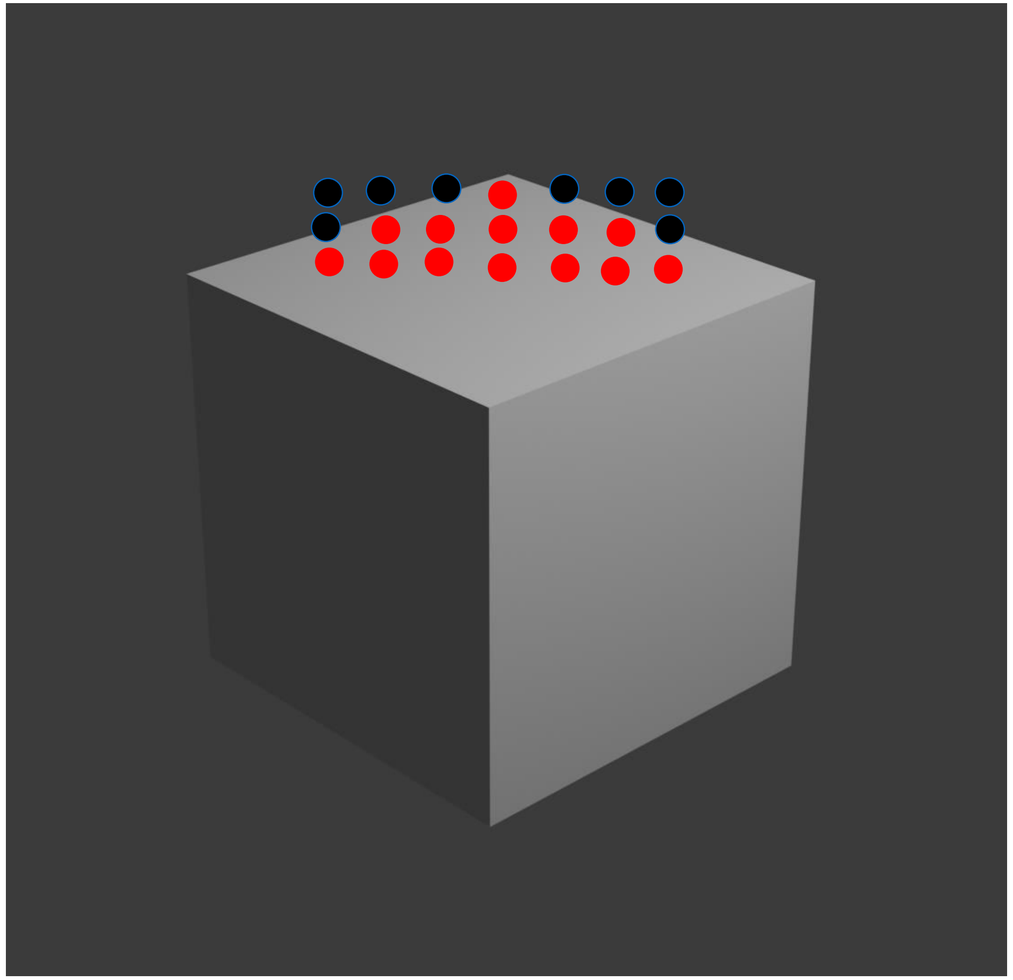
Raster-based Rendering

- The traditional rendering pipeline that uses projection to determine where 3D objects are placed on the 2D image.
- **For every polygon in my scene, what pixels can see it?**
- The EEVEE renderer is a raster-based renderer.



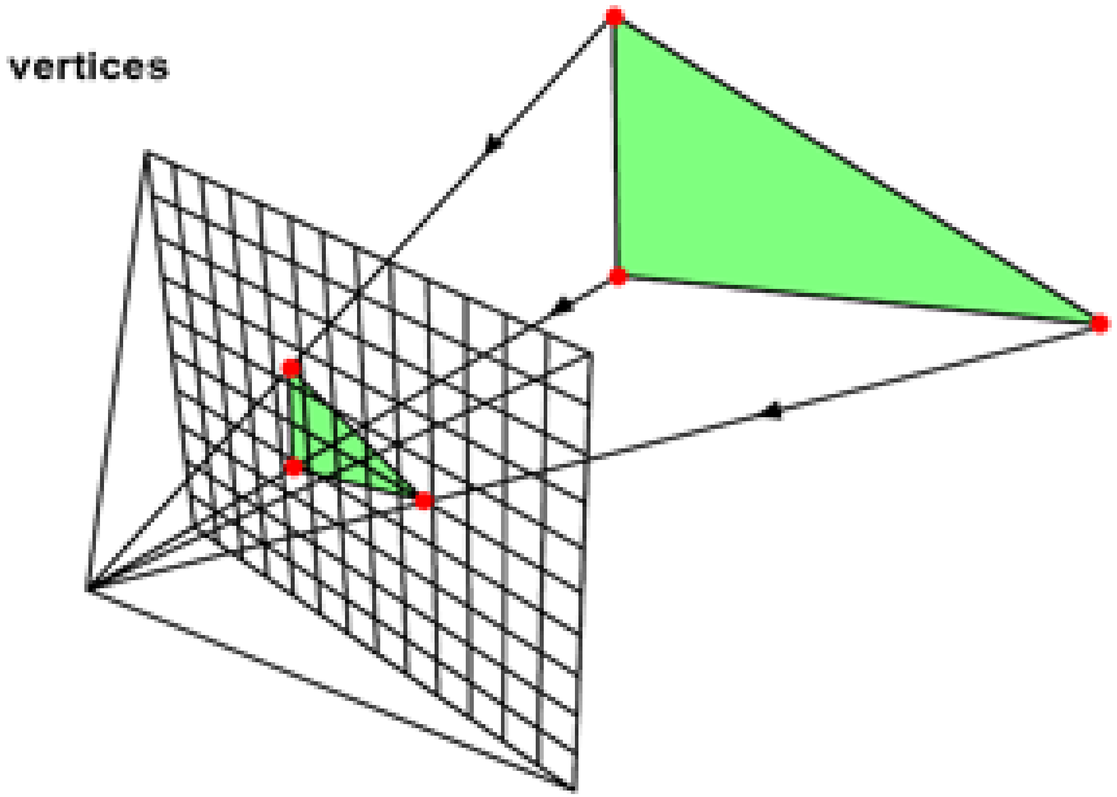
Ray Casting/Tracing

- A physics-based approach that considers rays of light that would enter into the camera.
- **For every pixel in my scene, what polygons can it see?**
  - What light sources can that polygon see?
- The Cycles renderer is a Ray Tracer.

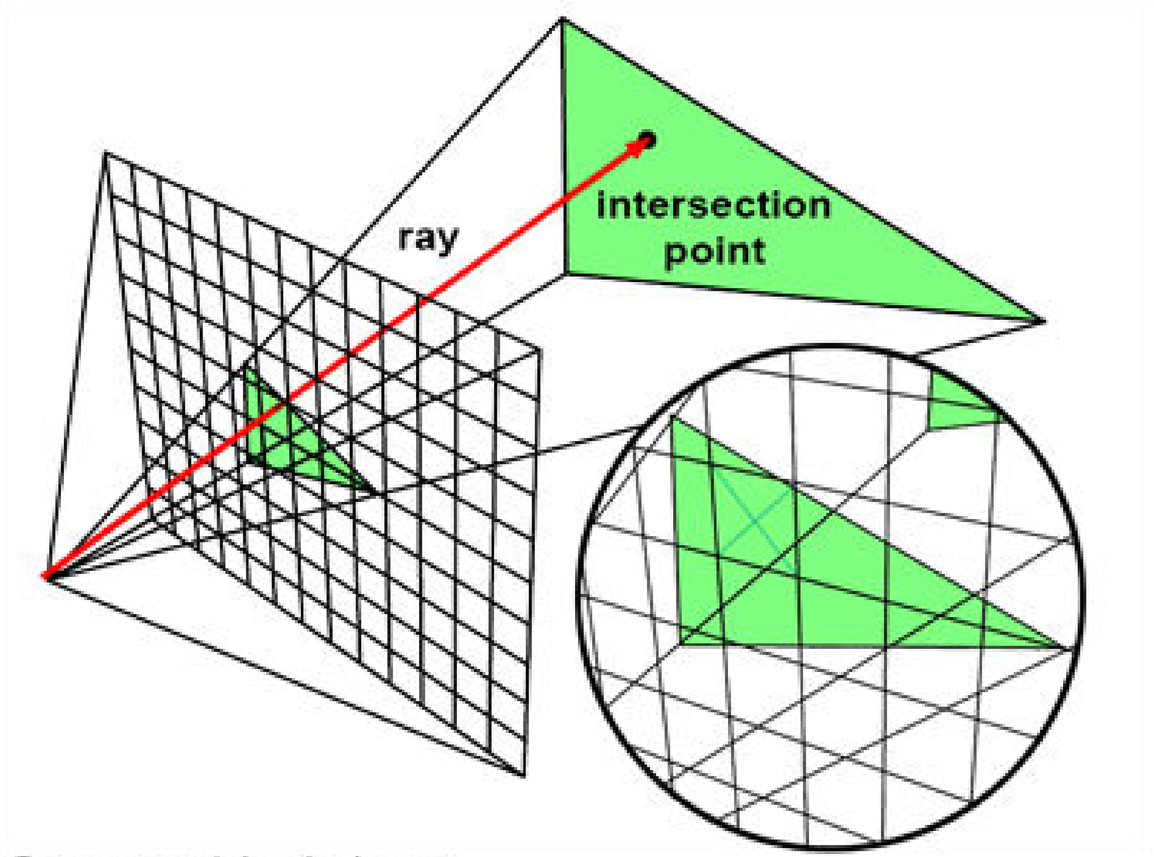


Comparison

Raster-based Rendering



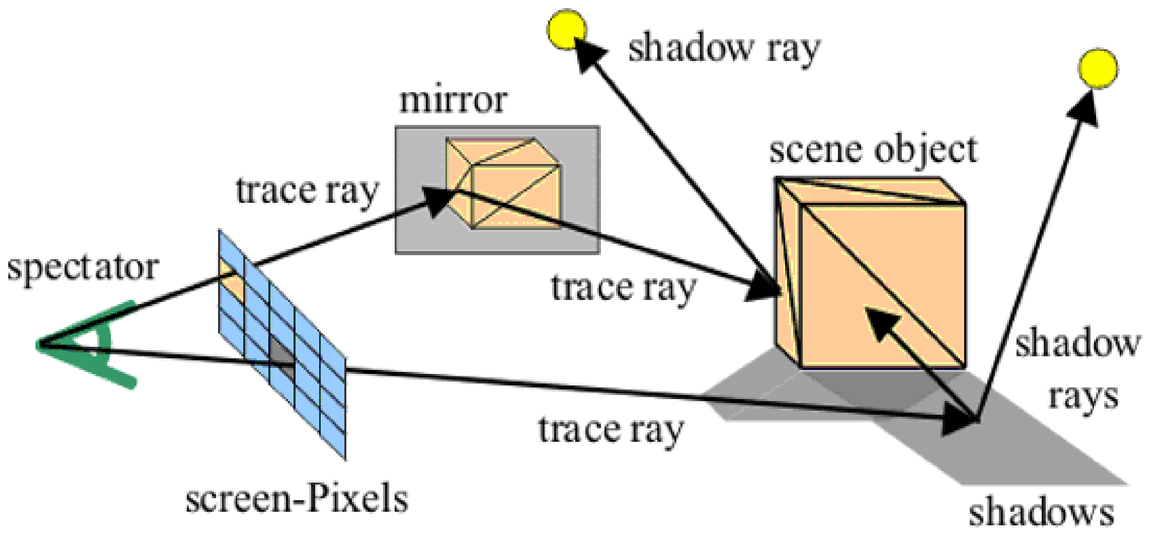
Ray Tracing



© www.scratchapixel.com

Ray Tracing

- Shoot a ray out through the pixel on the imaging plane
- When it hits something, ***recursively*** shoot more rays:
  - Towards light source(s) — direct lighting
  - Reflection ray (if applicable)
  - Refraction ray (if applicable)



Turner Whitted, 1980

Example Ray Traced Image

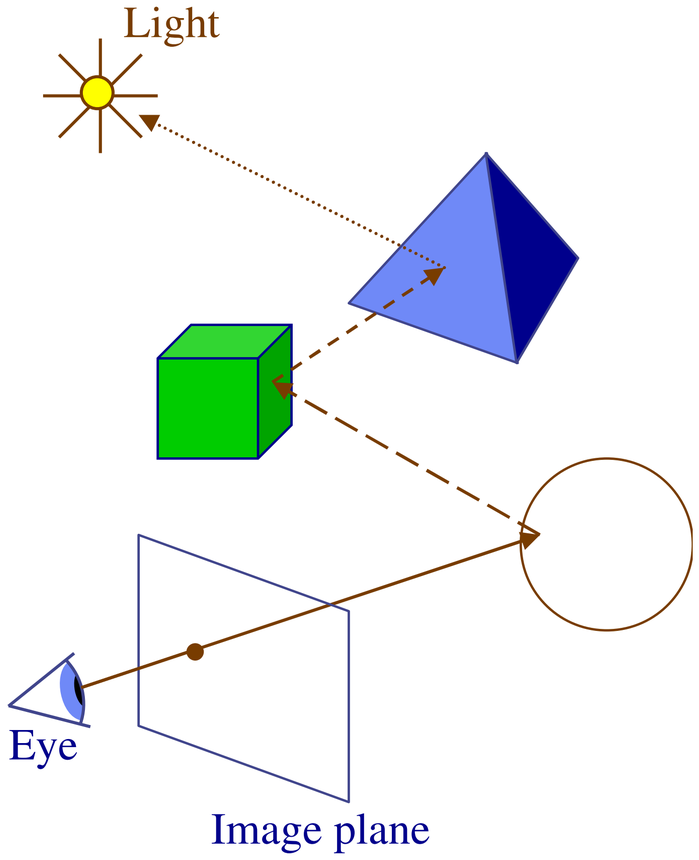
Example Ray Traced Image

Why is ray tracing slow?

- In rasterization-based rendering, you only need to loop through every object once.

- In raytracing, you need to loop through every object **at least** once **for every pixel.**

  - Rays also bounce and hit other objects!



Comparison

Raster-based Rendering

- **Pros**
  - Fast!
  - Default for libraries like OpenGL
  - Sufficient for most applications
- **Cons**
  - Lower quality renders

Ray Tracing

- **Pros**
  - Photorealistic quality!
- **Cons**
  - Slow!
    - …but GPUs and AI are making it more reasonable, even real-time!

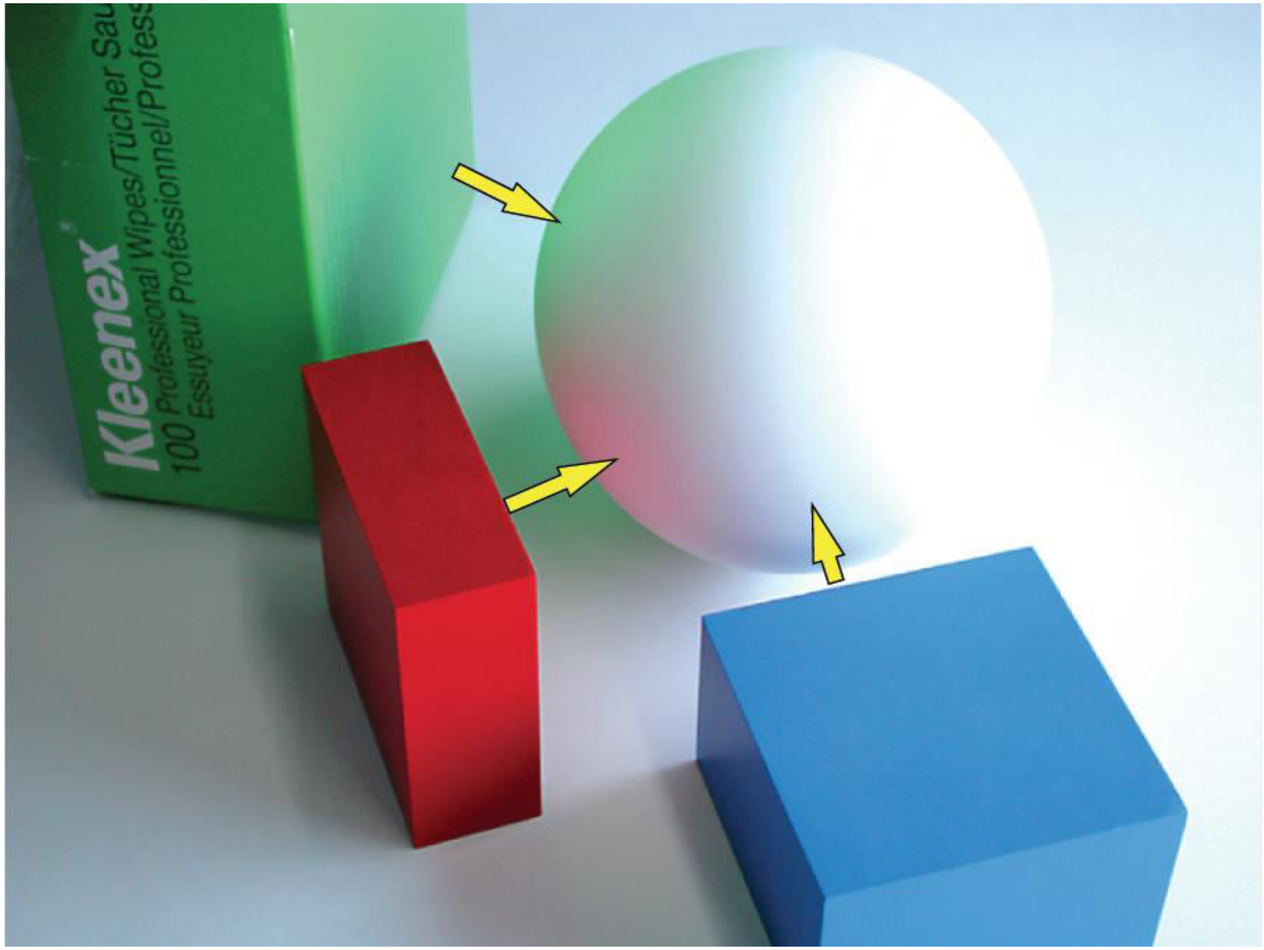
An Alternate Approach: Radiosity

- Math involves integrals over *all angles* of light coming *into* and *off* of a surface
- Approximated using *importance sampling*
- Effectively traces lots and lots of rays — compute intensive

Radiosity

- Because radiosity involves diffuse light, it is independent of the viewpoint
- Can precompute ahead of time if objects and light sources *do not change*
- But if something other than the camera moves… start over.
- Great for diffuse interreflections but has fallen out favor as ray tracers have gotten faster.

Diffuse Interreflections



Aside: University of Utah

University of Utah, 1972

Aside: University of Utah

Notable Alumni

- **James Blinn, Ph.D., 1978 —** Created specular lighting models, bump mapping and environment mapping for surface textures in graphical images.
- **Bui Tuong Phong, Ph.D., 1975** - Created the Phong shading and reflection models
- **Ed Catmull, Ph.D., 1974 —** Pioneer in computer animation who co-developed RenderMan rendering software. Co-founder of Pixar Animation Studios and winner of five Academy Awards.
- **Ivan Sutherland —** Inventor of Sketchpad, the first interactive graphics program with geometric constraints. Co-founded Evans & Sutherland with David Evans.

https://attheu.utah.edu/facultystaff/utahs-graphics-pioneers/

Demo: Interior Scene

- EEVEE vs Cycles
- Render Settings
- Speeding Up Renders
- Saving Outputs

Quick Tips: Faster Rendering in Cycles

- Use GPU as device if you have one and change the tile size to 512.
- Decrease the Max Samples or set a Time Limit.
- Under Light Paths, decrease the total Max Bounces to ~3.
- On complex 3D models, run the Decimate modifier to decrease the number of faces.

Don’t go too low on any of these settings or your scene will no longer look realistic.

Project 2

Blender Add-ons

- Add-ons can be helpful for speeding up the 3D modeling process.
- These can be installed using Edit -> Preferences -> Add-ons
- I recommend two add-ons:
  - Archimesh: Gives basic meshes for interior scene objects (doors, windows, etc.)
  - BlenderKit: Allows for quick import of free 3D models and textures.
    - Must be installed from website.

Animation and Simulation

CSCI 3800/6820: Computer Graphics

Outline

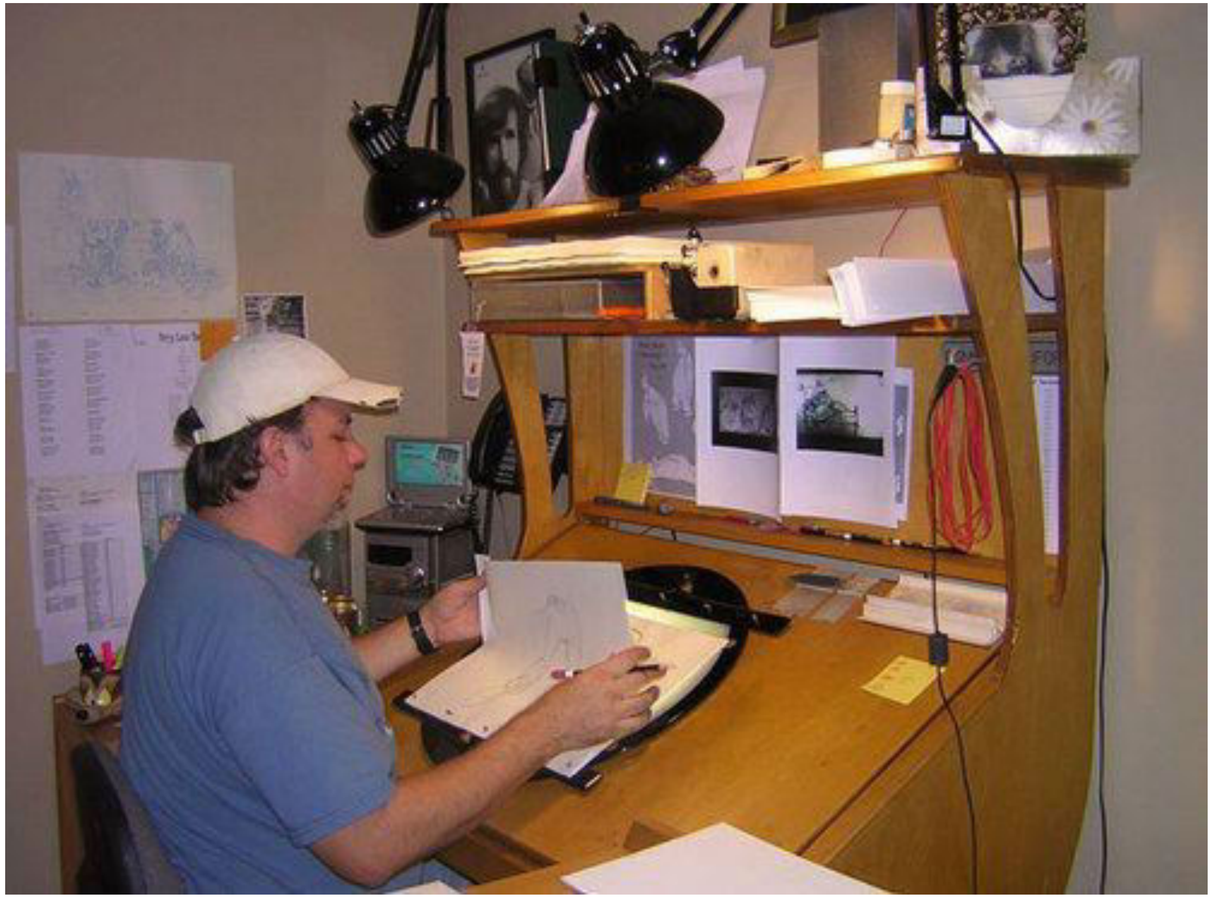
- Animation
  - Key Frames
  - Rigging
- Simulation
- Other Features in Blender

Animation

- **Animation** – A generated sequence of images that gives the appearance of movement.

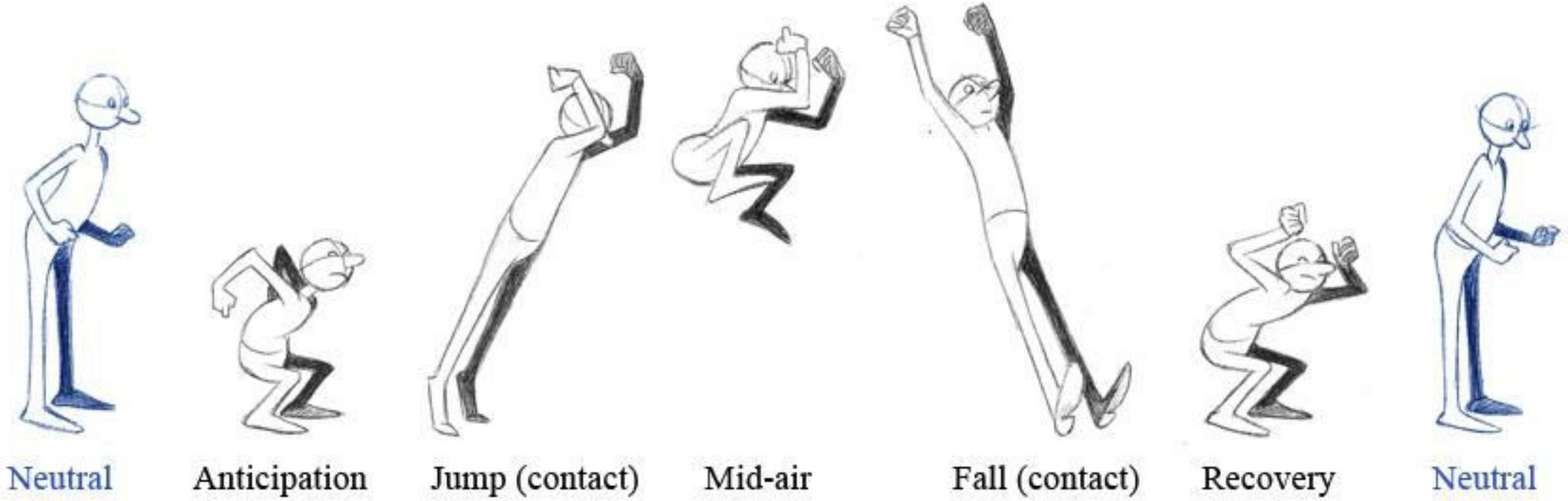
- For computer animation, it is generally specified as a set of keyframes that the system interpolates between.

Aside: Traditional Animation



Key Frames

- **Key Frames** - Specific locations/orientations of objects at designated times.



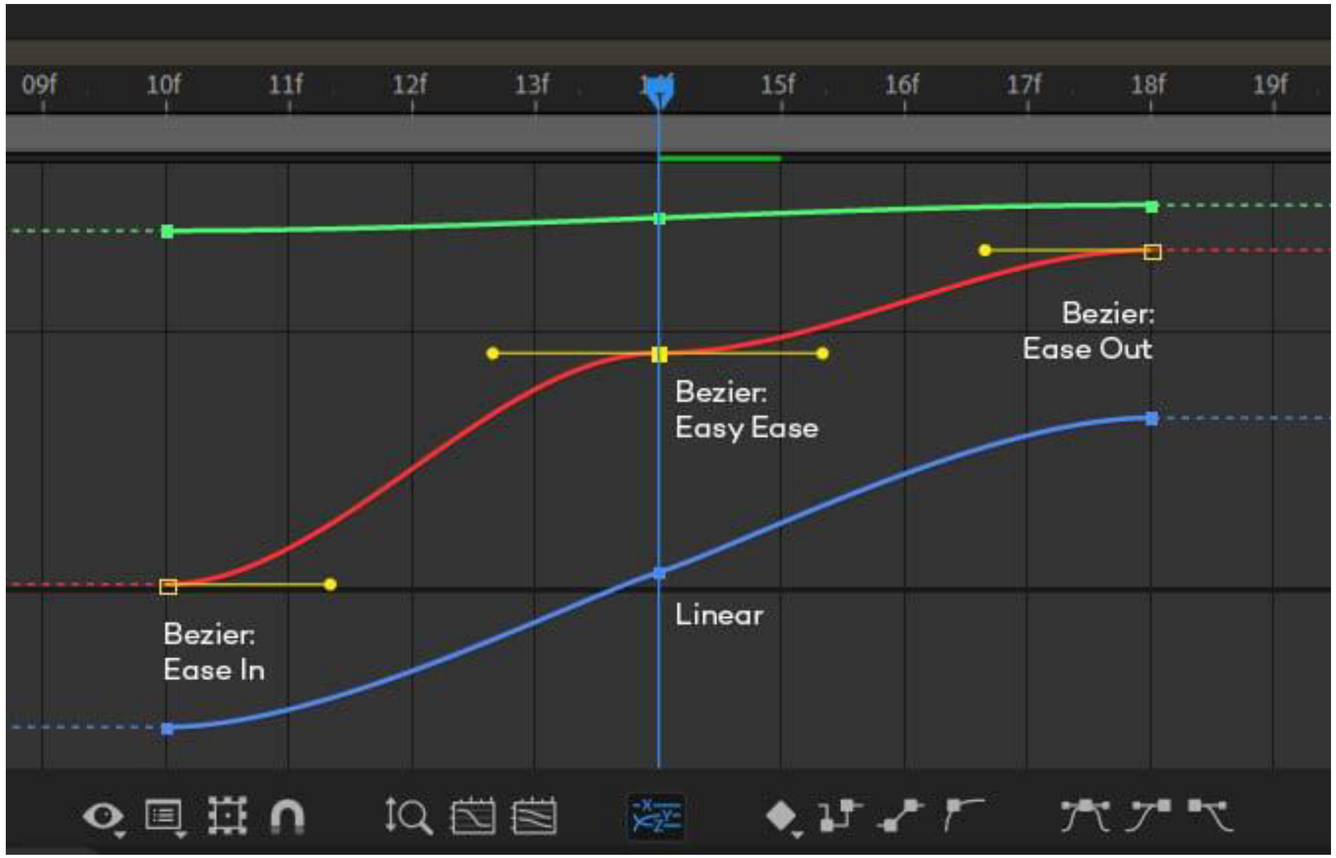
Key Frames

- **Key Frames** - Specific locations/orientations of objects at designated times.

https://wallawallastudio.com/article/rethinking-motion-capture/

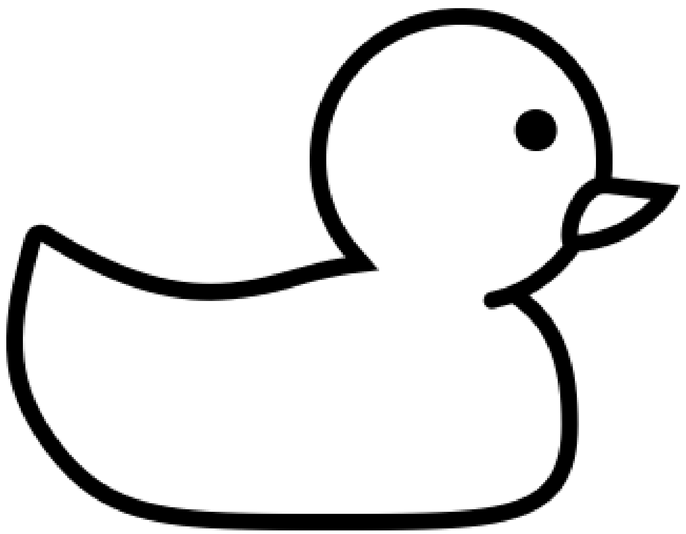
Key Frame Interpolation

- By default, most systems assume linear interpolation between key frames.
- It is also often possible to specify a curve that represents the interpolation.



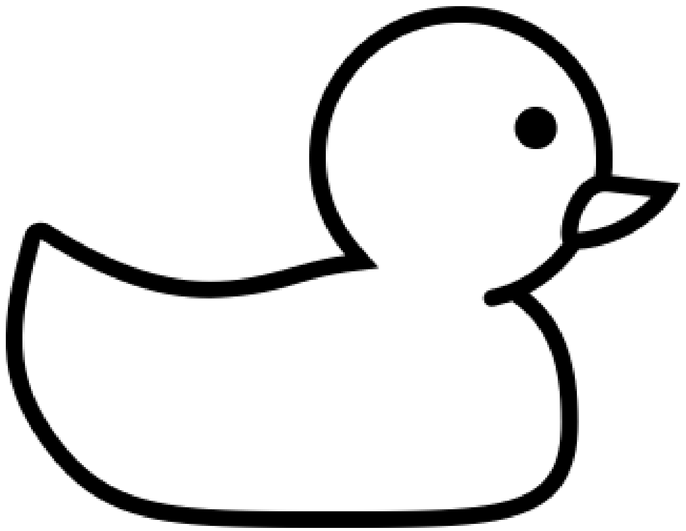
https://support.loupedeck.com/keyframe-interpolation

Key Frame Interpolation



Linear Interpolation

Key Frame Interpolation



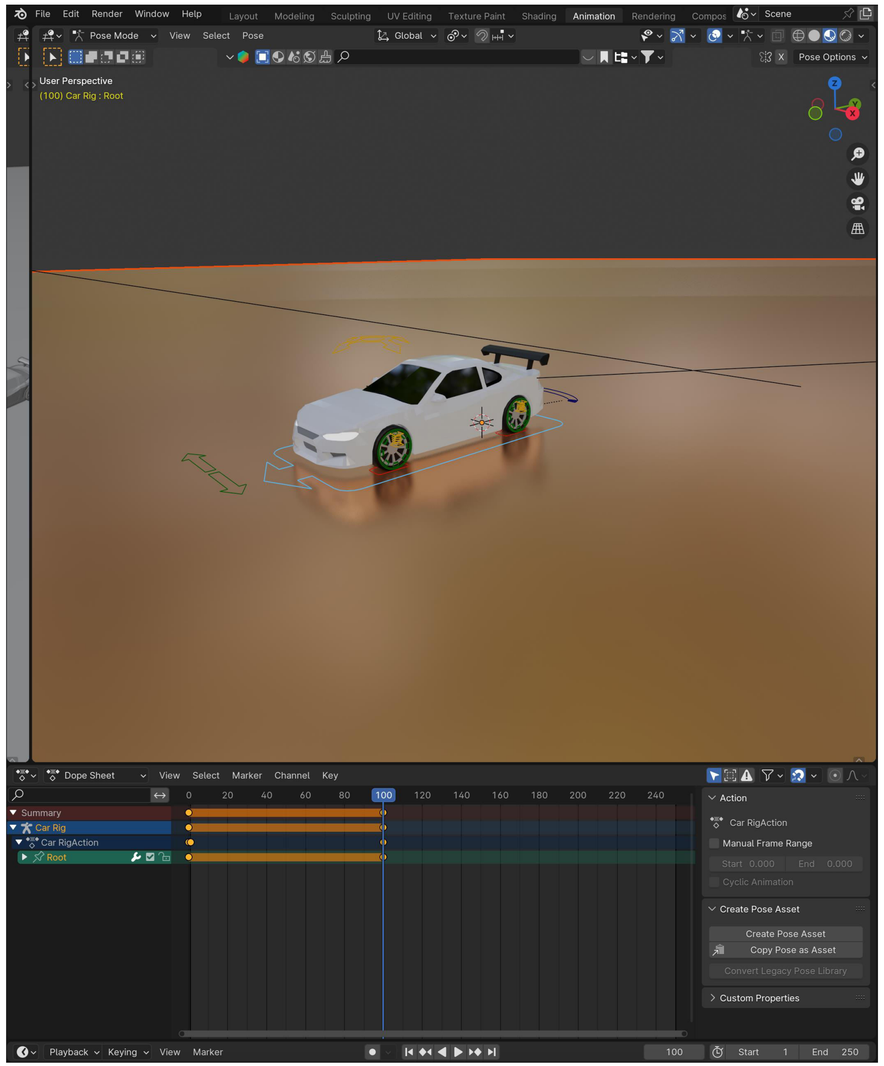
Ease In/Ease Out Interpolation

Key Frames in Blender

- You can specify the location, rotation, and scale of an object (or part of an object) at any time in the sequence using the “I” hotkey.

- Blender interpolates between these location, rotation, and scales for the other frames in the sequence.

- Press “Space” to play back your animation sequence.

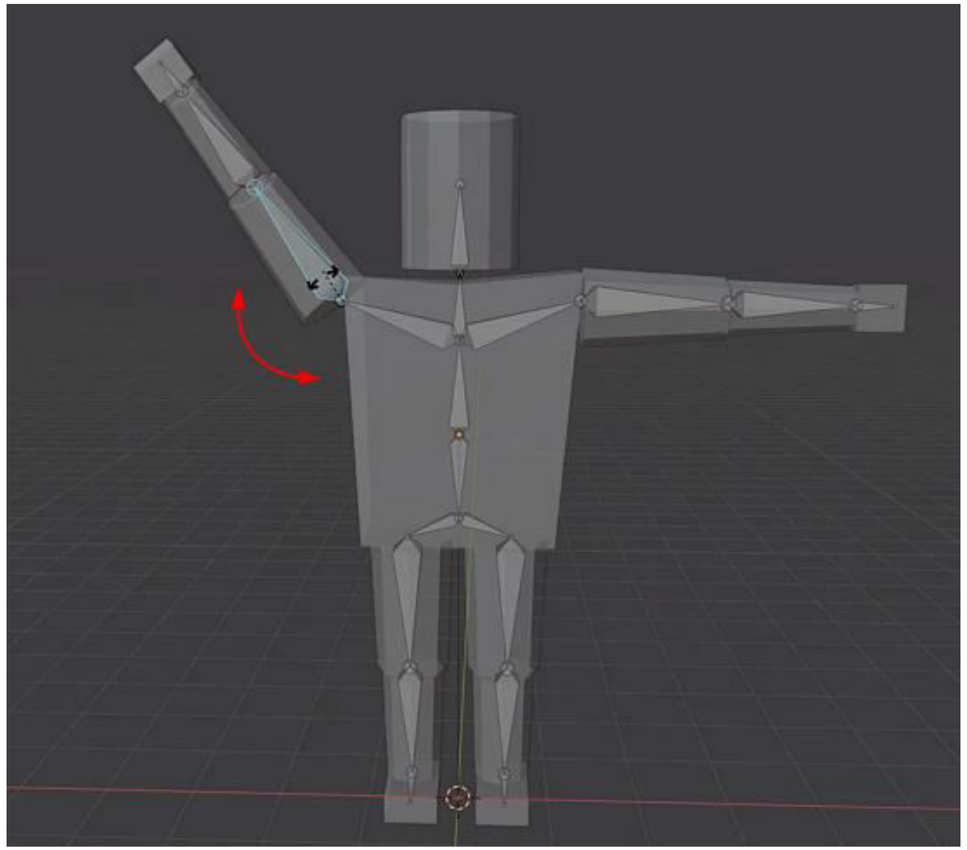


Demo: Rubber Duck

- Inserting Key Frames
- Animation Playback
- Changing Export Settings

Rigging

- **Rigging** is the process of making a “skeleton” that allows a specific part of the object to be moved independently.
- Think of it like creating a puppet out of your 3D model.
- Rigging is available in Blender, but it is a complex topic that is beyond the scope of this course.

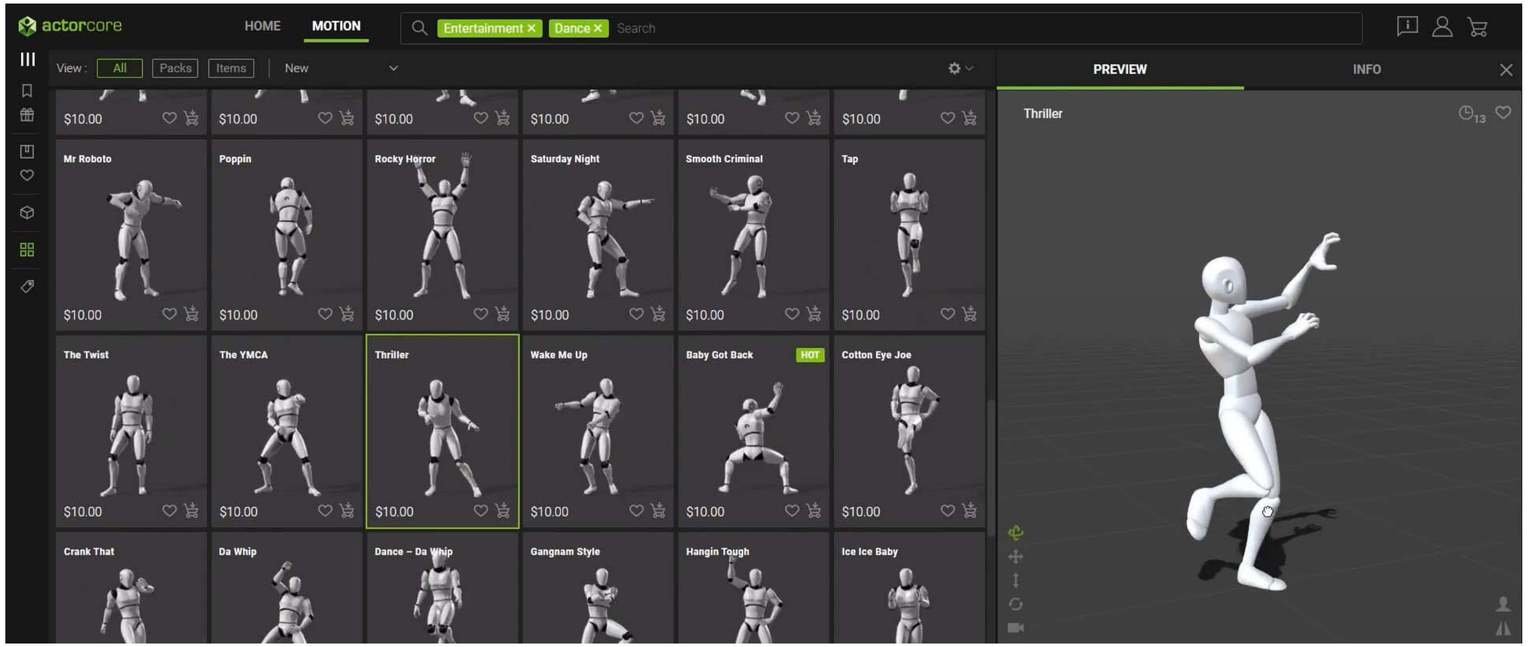


https://gamedevacademy.org/blender-rigging-tutorial/#Rigging_%E2%80%93_Part_1

Animation Sequences

- If you are creating 3D characters for games, you can save multiple sequences of animation for different actions.

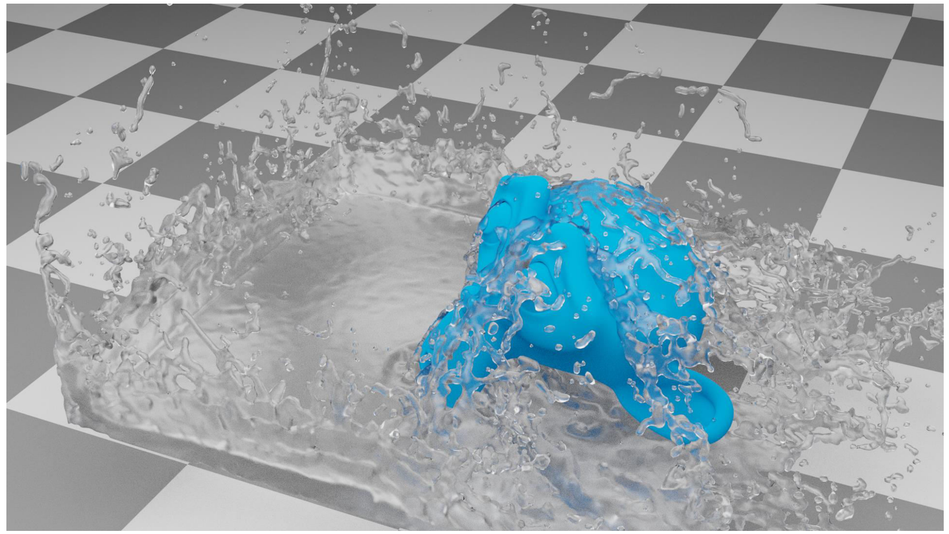
- Examples include:
  - Walking
  - Running
  - Falling



https://magazine.reallusion.com/2021/01/01/3d-motions-for-2d-animation-production-now-a-reality/

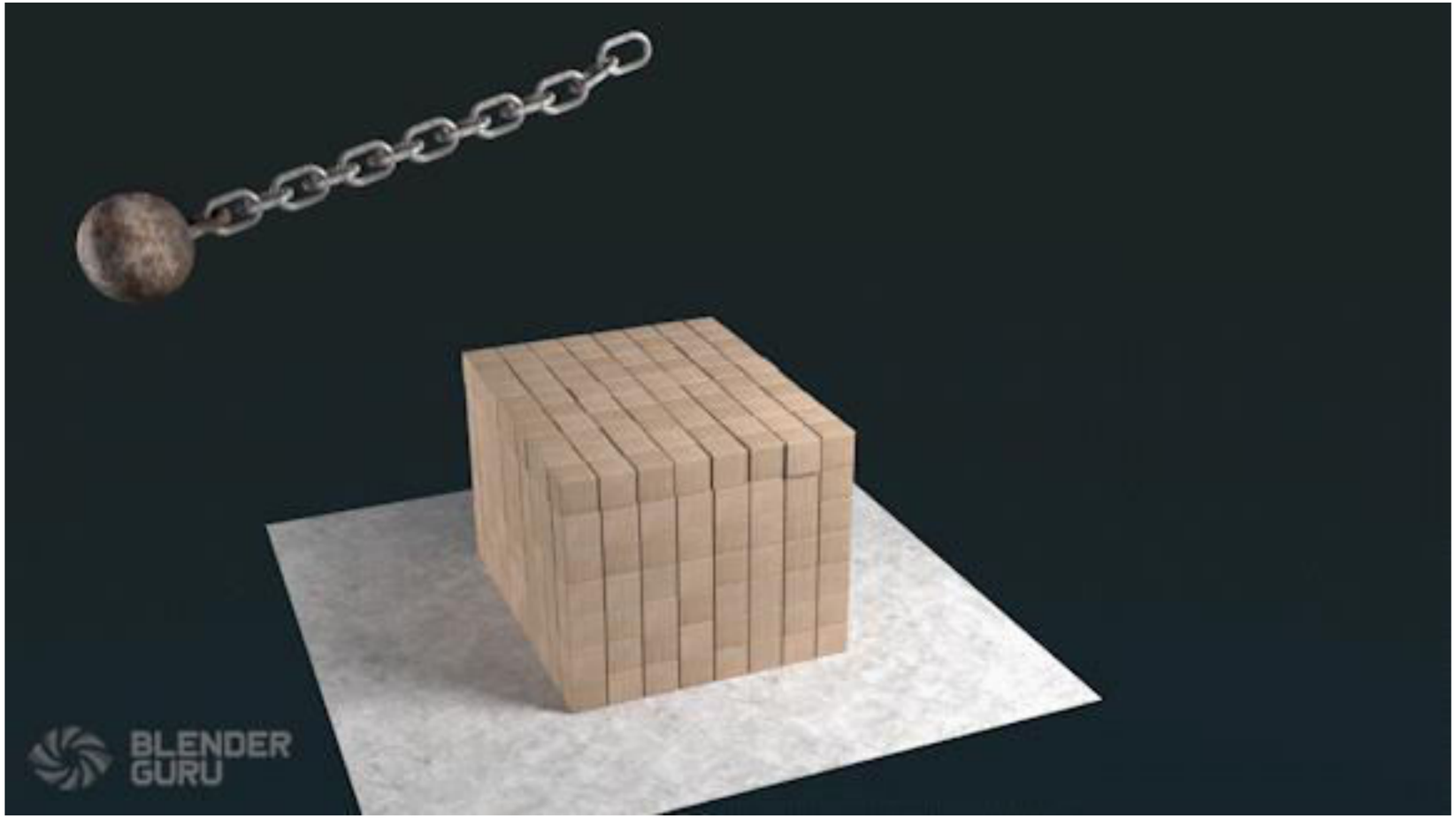
Simulation

- Objects can be also be animated by simulating physical laws.
- Examples include:
  - Fluids: Liquids, Smoke, Clouds, Fire
  - Rigid Bodies: Sand, Snow, Gravel
  - Deformations: Skin, Cloth, Hair, Fur



Fluid Simulation

Rigid Body Simulation



Deformation Simulation

Differential Equations

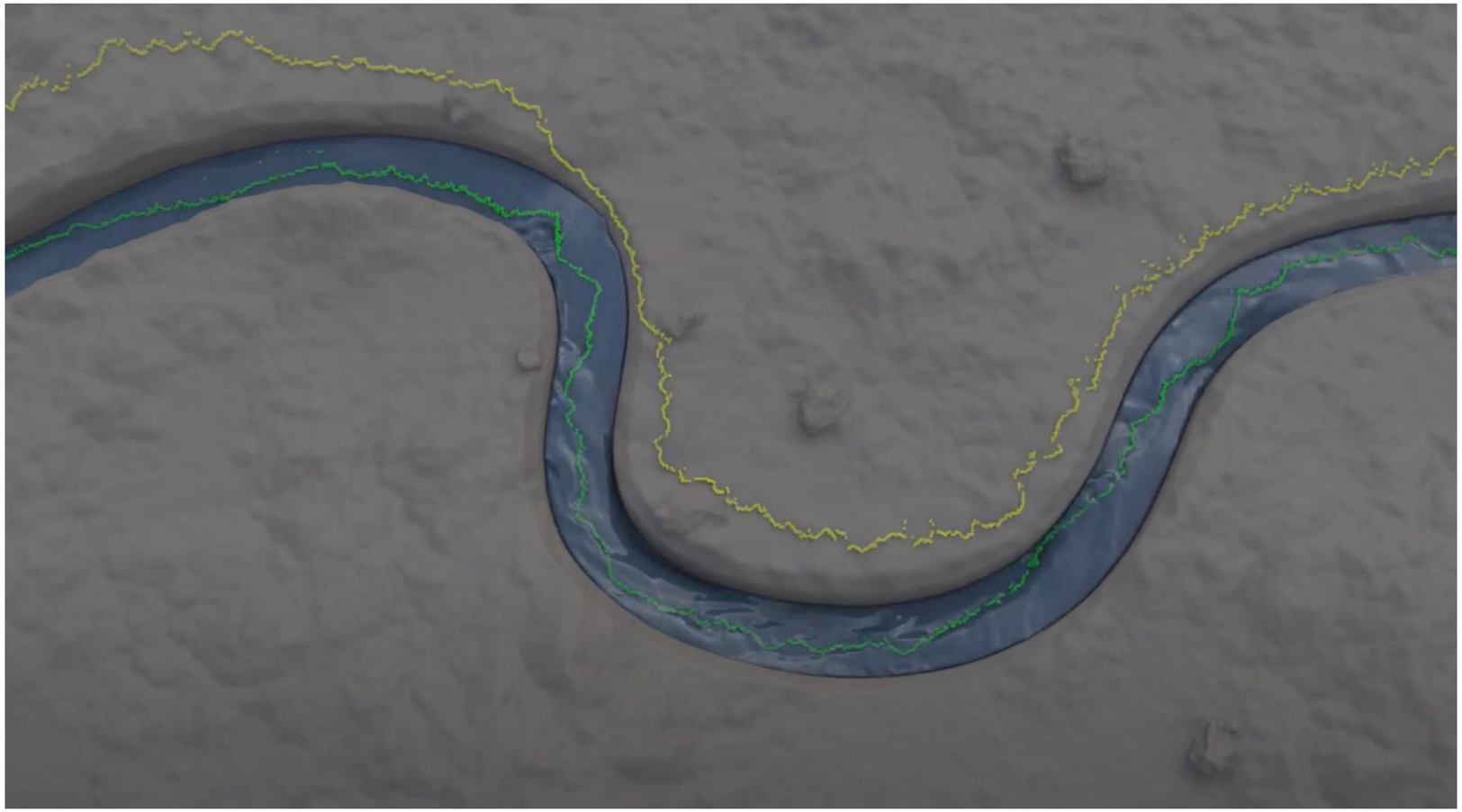
- Every kind of simulation can be described as a set of governing differential equations.

- For example, fluid simulation uses the famous Navier-Stokes equations:

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} - \nu \nabla^2 \mathbf{u} = -\frac{1}{\rho}\nabla p + \mathbf{g}.$$

- These equations are beyond the scope of this course.

Dr. Hart’s Research: Fluid Carving



https://www.youtube.com/watch?v=CWrTJ7ikQzw

Aside: Way of the Water

Simulation in Blender

- Simulations can be described using the Physics tab in the right options bar.
- Use Rigid Body to describe objects that interact with each other.
  - Use **active** to describe an object (it will fall due to gravity).
  - Use **passive** to describe a barrier to an object (it won’t fall).
- Change collision shape to **Mesh** for best results

Demo: Too Many Ducks

- Simulations
- Active vs Passive Objects
- Changing Weights
- “Scrubbing” the simulation

What are these other tabs…

Compositing

- The Compositing tab allows you to post-process the images generated from the render.
- Generally has two purposes:
  - To change the colors for stylistic effects (think Adobe Lightroom)
  - To combine with recorded footage for special effects
- Composition is beyond the scope of this course (take Computer Vision!)

Compositing

Geometry Nodes/Scripting

- The **Geometry Nodes** and **Scripting** tabs allow you to make modifications to your geometry using mathematical or coding based descriptions.

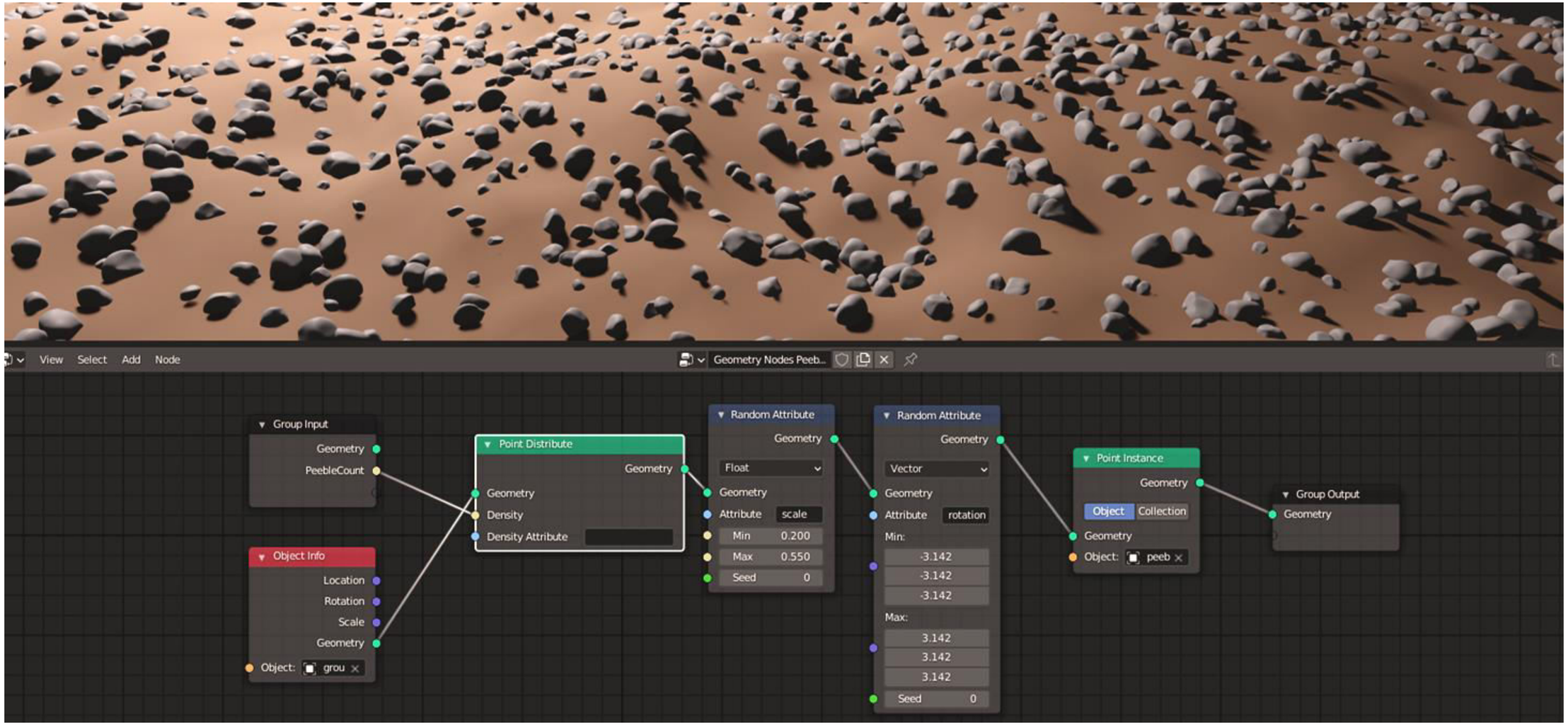
- Examples include

  - Random perturbations to a surface

  - Scattering points along a surface

- Geometry Nodes are beyond the scope of this course.

Geometry Nodes



https://www.blendernation.com/2020/12/18/blender-2-92-new-feature-geometry-nodes/

Keep Exploring!

- This marks the end of the Blender unit of the course, but we have just scratched the surface of available features.
- I hope you will keep exploring what you can do with this amazing software and create something new.
- There are plenty of tutorials on Youtube and other places where you can continue to learn.
- Next, we move on to some theory…

Vectors and Matrices

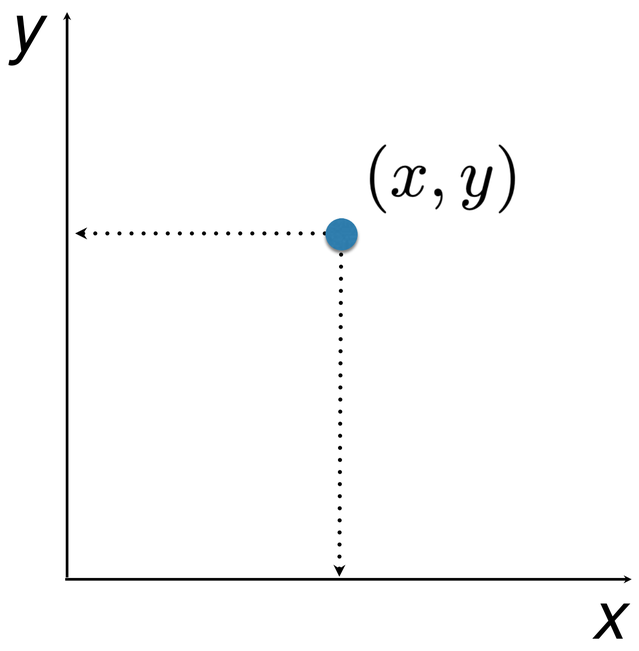
CSCI 3800/6820: Computer Graphics

In this course, we are going to talk a lot about vectors and matrices…

…and I mean a lot.

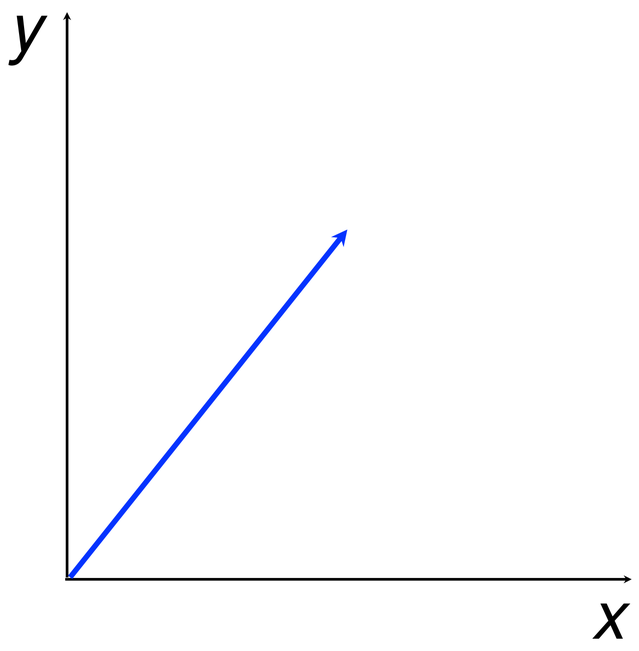
So let’s make sure we have the basic math down.

Points



Points can be described by their Cartesian coordinates.
Coding: use short arrays or n-tuples

Vectors



Used to describe a
direction and/or magnitude

Vectors can also be thought of
in terms of Cartesian coordinates

Vectors

$$\mathbf{a} = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \qquad \mathbf{b} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \qquad \mathbf{c} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}$$

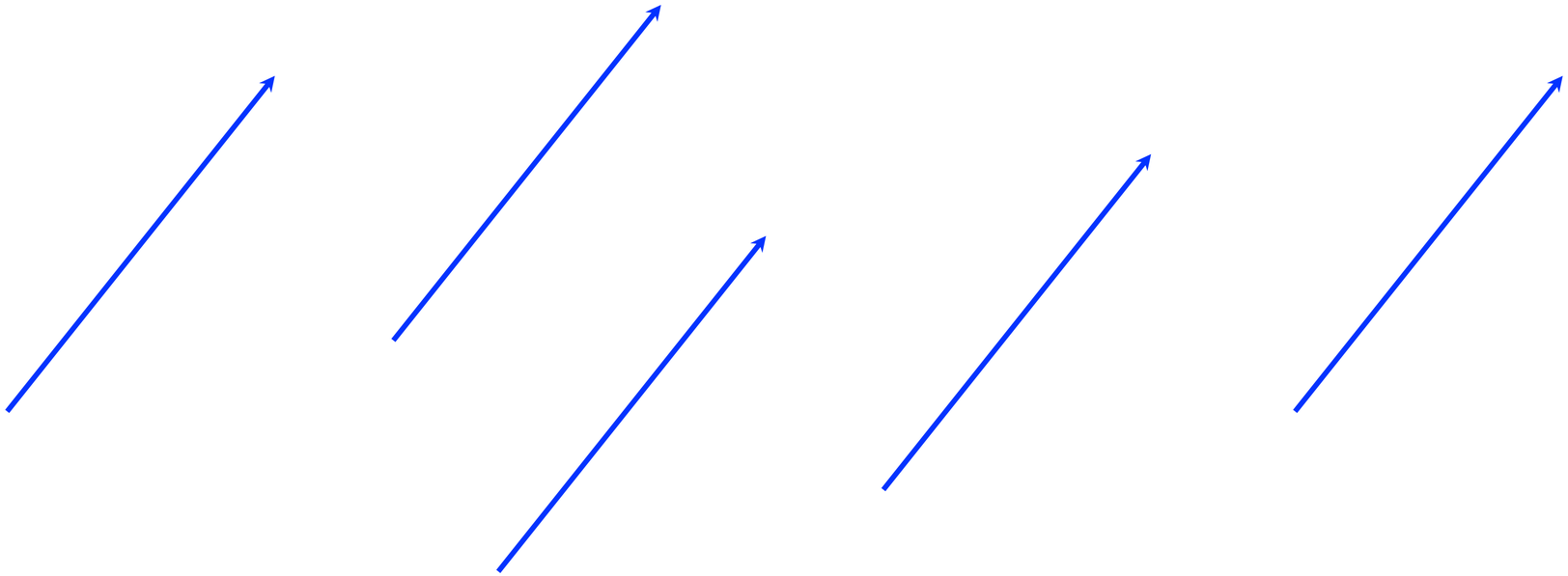
Coding: just store in an array

Vectors

$$\mathbf{a} = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix}$$

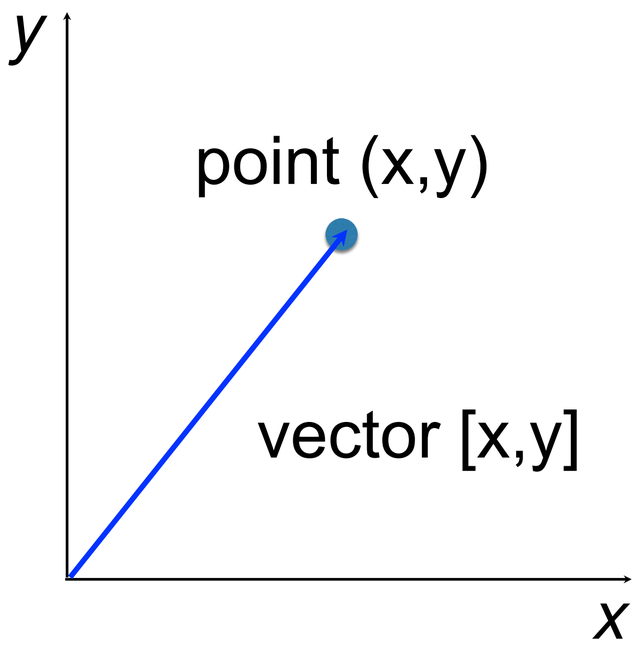
Often use subscripts to denote elements of a vector.
Coding: use `a[i]`

Vectors



Vectors are directional quantities
without a specific location

Points and Vectors



Points and vectors are different but related

Interchangeable (but be careful)

Matrices

$$\mathbf{M} = \begin{bmatrix} 3 & 1 & 8 & 5 \\ -1 & 4 & -3 & 3 \\ 2 & 0 & -1 & 4 \end{bmatrix}$$

A matrix is an n by m array of numbers

Coding: Use a numpy array with multiple []
Example: np.array([[1,2],[3,4]])

Notation

$$\mathbf{M} = \begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{bmatrix}$$

Index an array by row, then column

Coding: Use a[i,j]

Square Matrices

$$\mathbf{M} = \begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{bmatrix}$$

Diagonal

Square matrices have the same number of rows as columns (n = m)

If everything off the diagonal is 0, it is a diagonal matrix

Vectors as Matrices

$$\mathbf{v} = \begin{bmatrix} 2 \\ 3 \\ -1 \end{bmatrix}$$

A vector is simply an n x 1 matrix

(technically, this is a *column vector*—some use *row vectors*)

Linear Algebra in this Course

Essential

Vectors

- Magnitude
- Unit Vectors
- Normalization
- Dot Products
- Orthogonal Vectors
- Cross Products

Matrices

- Matrix Multiplication
- Stacked Vectors

Helpful, but Nonvital

Vectors

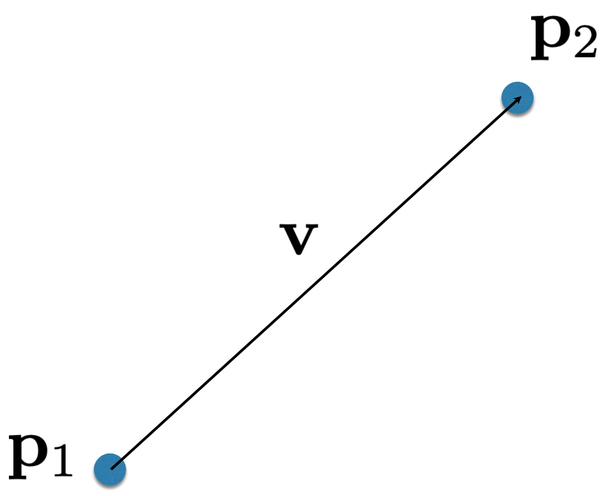
- Vector Addition
- Commutative Properties
- Vector Angles

Matrices

- Matrix Transpose
- Determinant
- Matrix Rank
- Orthogonal Matrices
- Matrix Inversion

Additional slides available online

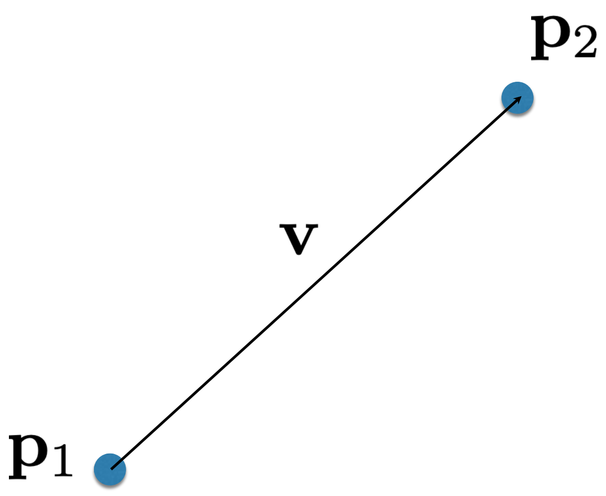
Vectors as Displacements



$$\mathbf{p}_1 + \mathbf{v} = \mathbf{p}_2$$

Type system: point + vector = point

Vectors as Displacements

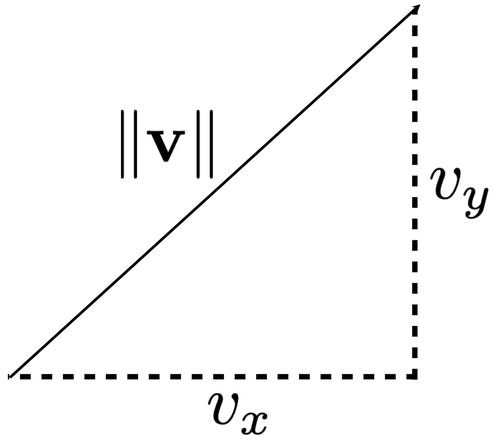


$$\mathbf{v} = \mathbf{p}_2 - \mathbf{p}_1$$

Type system: point - point = vector

Magnitude

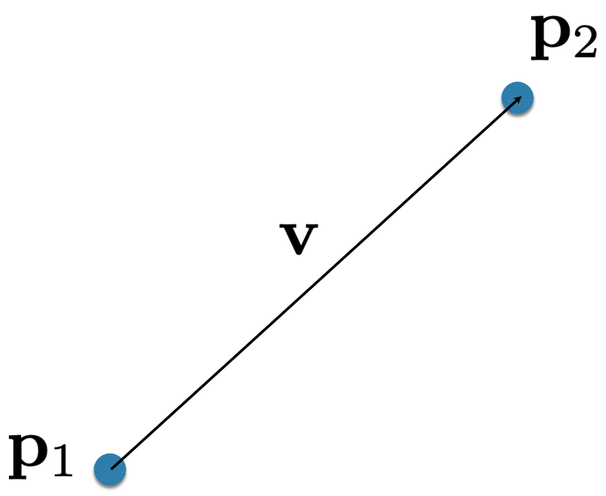
- The magnitude (length) of a vector can be calculated using Pythagorean theorem
- Sometimes called the *norm* of the vector
- Note: there are other vector norms, but assume this unless stated otherwise



$$\|\mathbf{v}\| = \sqrt{v_1^2 + v_2^2 + \cdots + v_n^2}$$

"magnitude", "length", or "norm" of $\mathbf{v}$

Distance



$$\|\mathbf{v}\| = \|\mathbf{p}_2 - \mathbf{p}_1\| = \sqrt{(p_2[x] - p_1[x])^2 + (p_2[y] - p_1[y])^2}$$

Convenient way to write / calculate distance between points

Example

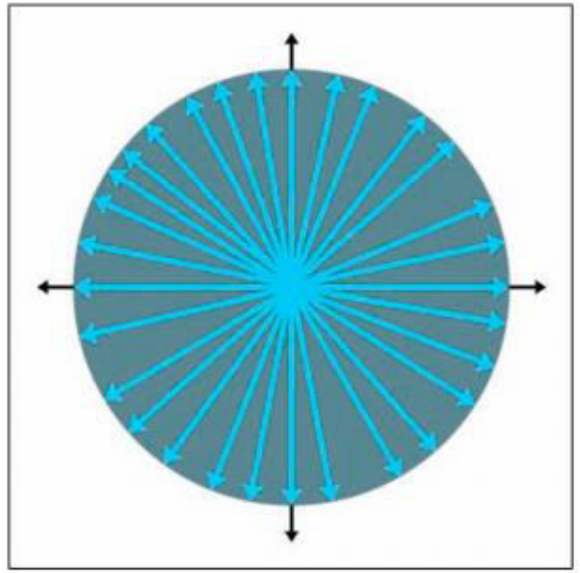
$$a = \begin{bmatrix} -5.1 \\ 7.7 \end{bmatrix}$$

$$\|a\| = \sqrt{(-5.1)^2 + (7.7)^2} = 9.236$$

Unit Vectors

$$\|v\| = 1$$

- A “unit vector” has a **length of one**
- Useful to describe <u>direction</u> when we don’t care about magnitude



Normalizing

- Sometimes we want to **normalize** a vector to have the same direction but unit length
- Key: just divide it by its own length
- Can’t do this for the zero vector

$$\hat{\mathbf{v}} = \frac{\mathbf{v}}{\|\mathbf{v}\|} = \frac{\mathbf{v}}{\sqrt{v_1^2 + v_2^2 + \cdots + v_n^2}}$$

Example

$$a = \begin{bmatrix} -5.1 \\ 7.7 \end{bmatrix} \qquad \|a\| = 9.236$$

$$\hat{a} = \frac{a}{\|a\|} = \begin{bmatrix} -5.1/9.236 \\ 7.7/9.236 \end{bmatrix} = \begin{bmatrix} -0.55 \\ 0.83 \end{bmatrix}$$

Vector Dot Products

$$\mathbf{a} \cdot \mathbf{b} = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} \cdot \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} = \sum_{i=1}^{n} a_i b_i$$

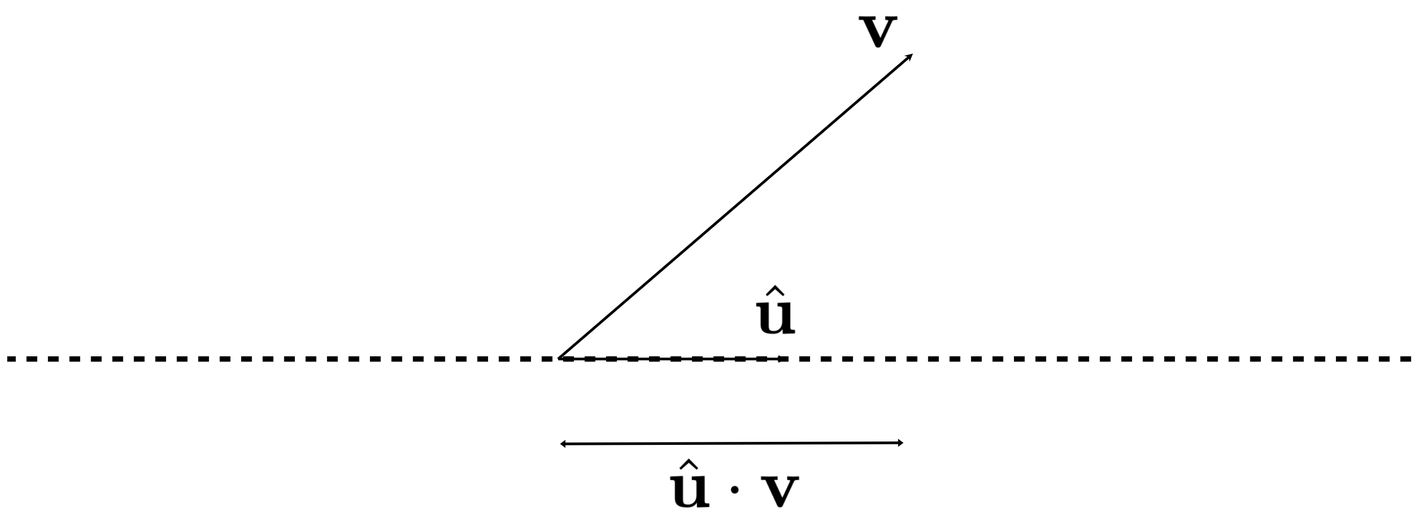
One of the most commonly used
vector operations in graphics

Example

$$a = \begin{bmatrix} -5.1 \\ 7.7 \end{bmatrix} \qquad b = \begin{bmatrix} 3.1 \\ 4.5 \end{bmatrix}$$

$$a \cdot b = (-5.1)(3.1) + (7.7)(4.5) = 18.84$$

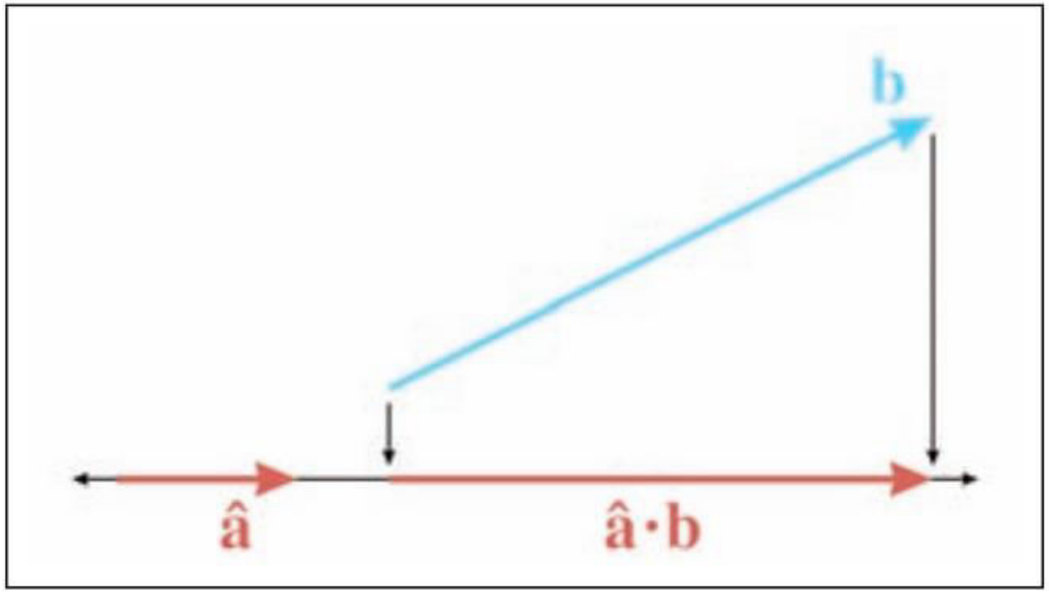
Geometric Interpretation



The dot product of a vector and a <u>unit</u> vector
is the length of the projection onto that unit vector

“How much of this vector lies in that direction?”

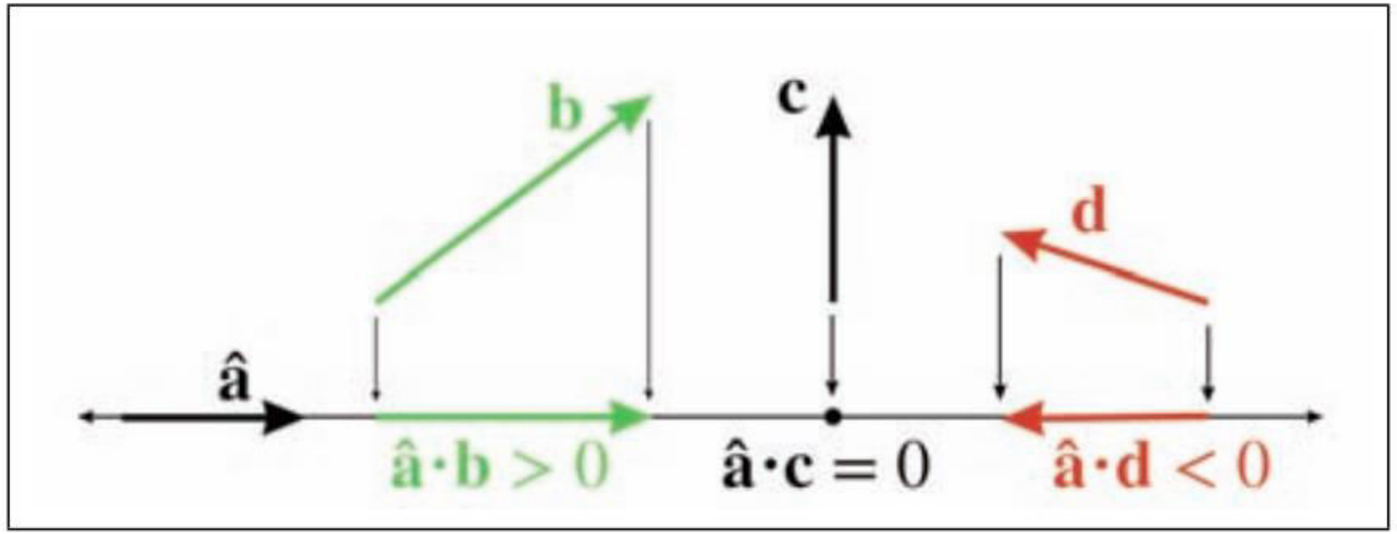
Geometric Interpretation



The dot product of a vector and a <u>unit</u> vector
is the length of the projection onto that unit vector

“How much of this vector lies in that direction?”

Sign of the Dot Product



The sign of the dot product between two vectors tells whether the projection is in the same direction

Dot Product Interactives

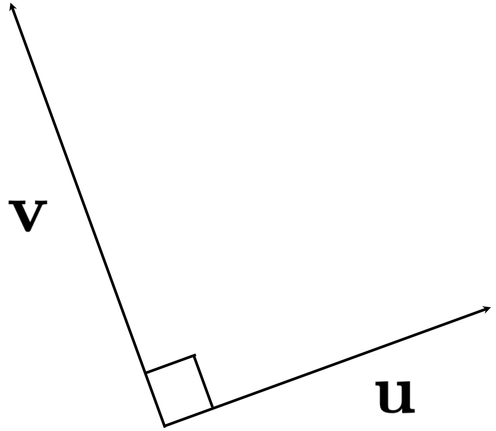
https://www.learndatasci.com/glossary/dot-product/

https://www.falstad.com/dotproduct/

https://www.geogebra.org/m/Yu6869By

Orthogonality

- Vectors whose dot product is zero are said to be “orthogonal”
- “Right angle” to each other (regardless of length)
- The zero vector is trivially orthogonal to everything else



$$\mathbf{u} \cdot \mathbf{v} = 0$$

Lengths

$$\mathbf{v} \cdot \mathbf{v} = \|\mathbf{v}\|^2$$

The dot product of something with itself
is its own length squared

Tip: lots of “distance” tests only need squared distance

Cross Product

$$\begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} \times \begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix} = \begin{bmatrix} y_1 z_2 - z_1 y_2 \\ z_1 x_2 - x_1 z_2 \\ x_1 y_2 - y_1 x_2 \end{bmatrix}$$

Result is a vector

Only done in 3D

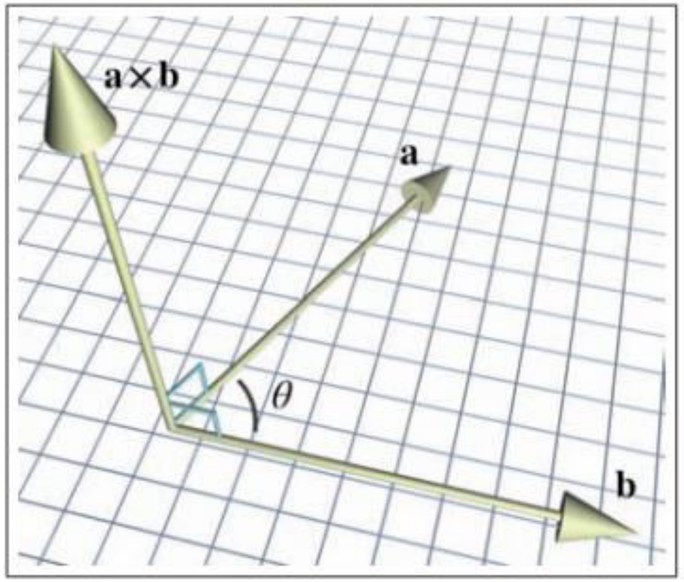
Example

$$\begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} \times \begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix} = \begin{bmatrix} y_1 z_2 - z_1 y_2 \\ z_1 x_2 - x_1 z_2 \\ x_1 y_2 - y_1 x_2 \end{bmatrix}$$

$$a = \begin{bmatrix} 2.3 \\ -1.4 \\ 3.7 \end{bmatrix} \qquad b = \begin{bmatrix} -4.5 \\ 6.2 \\ 1.1 \end{bmatrix}$$

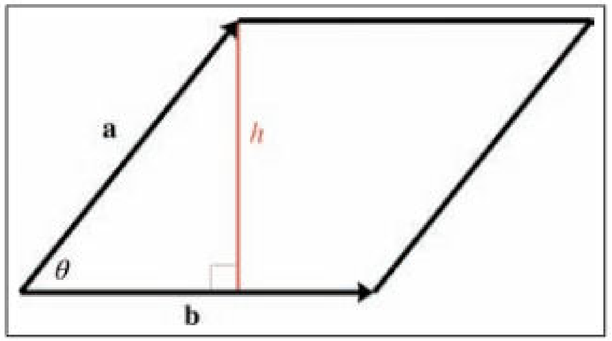
$$a \times b = \begin{bmatrix} (-1.4)(1.1) - (3.7)(6.2) \\ (3.7)(-4.5) - (2.3)(1.1) \\ (2.3)(6.2) - (-1.4)(-4.5) \end{bmatrix} = \begin{bmatrix} -24.48 \\ -19.18 \\ 7.96 \end{bmatrix}$$

Geometric Interpretation



The cross product of two vectors is
another vector orthogonal to the two
(really useful property in 3D geometry!)

Geometric Interpretation



The length of the cross product of two vectors
is the area of the parallelogram spanned by the two

Remembering the Cross Product

Let’s try it in NumPy…

Linear Algebra in this Course

Essential

Vectors

- Magnitude
- Unit Vectors
- Normalization
- Dot Products
- Orthogonal Vectors
- Cross Products

Matrices

- Matrix Multiplication
- Stacked Vectors

Helpful, but Nonvital

Vectors

- Vector Addition
- Commutative Properties
- Vector Angles

Matrices

- Matrix Transpose
- Determinant
- Matrix Rank
- Orthogonal Matrices
- Matrix Inversion

Additional slides available online

To remember matrices, let’s take some slides directly from discrete structures…

Matrix

- **Definition**: A *matrix* is a rectangular array of numbers. A matrix with *m* rows and *n* columns is called an $m \times n$ matrix.
- The plural of matrix is *matrices*.
- A matrix with the same number of rows and columns is called a *square* matrix.

$$\begin{bmatrix} 1 & 2 \\ 5 & 0 \\ 1 & 3 \end{bmatrix}$$

3 x 2
Matrix

Elements of a Matrix

In general, we can denote a matrix A such that

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ . & . & . & . \\ . & . & . & . \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}$$

Where a_{ij} is the number found at the ith row and jth column.

We call a_{ij} an *element* of the matrix.

Matrix Addition

- To add two matrices together, simply add the corresponding elements in each matrix. (We won't use this very often)

- In other words, to compute $A + B$, add every a_{ij} to every b_{ij}.

$$\begin{bmatrix} 1 & 0 & -1 \\ 2 & 2 & -3 \\ 3 & 4 & 0 \end{bmatrix} + \begin{bmatrix} 3 & 4 & -1 \\ 1 & -3 & 0 \\ -1 & 1 & 2 \end{bmatrix} = \begin{bmatrix} 4 & 4 & -2 \\ 3 & -1 & -3 \\ 2 & 5 & 2 \end{bmatrix}$$

A $\qquad$ B $\qquad$ $A + B$

Matrix Multiplication

- Matrix multiplication is a unique operation that is different than just multiplying corresponding elements.
- To compute a matrix multiplication, each element corresponds to a row's elements multiplied by a column's elements and added together.
- Mathematically, if $AB = C$, then add every a_{ij} to every b_{ij}.

$$c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + \ \ldots + a_{ik}b_{kj}$$

Visualizing Matrix Multiplication

$$B = \begin{bmatrix} b_{11} & a_{12} & \dots & b_{1j} & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & b_{2j} & \dots & b_{2n} \\ . & . & & . & & \\ . & . & & . & & \\ b_{k1} & b_{k2} & \dots & b_{kj} & \dots & b_{kn} \end{bmatrix}$$

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1k} \\ a_{21} & a_{22} & \dots & a_{2k} \\ . & . & & . \\ . & . & & . \\ a_{i1} & a_{i2} & \dots & a_{1k} \\ . & . & & . \\ . & . & & . \\ a_{m1} & a_{m2} & \dots & a_{mk} \end{bmatrix}$$

$$AB = \begin{bmatrix} c_{11} & c_{12} & \dots & c_{1n} \\ c_{21} & b_{22} & \dots & c_{2n} \\ . & . & & . \\ . & . & c_{ij} & . \\ . & . & & . \\ c_{m1} & c_{m2} & \dots & c_{mn} \end{bmatrix}$$

$$c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + \ \dots + a_{ik}b_{kj}$$

Example

$$B = \begin{bmatrix} -1 & 1 & 0 \\ 3 & 2 & -2 \end{bmatrix}$$

$$A = \begin{bmatrix} 1 & 2 \\ 5 & 0 \\ 1 & 3 \end{bmatrix} \quad AB = \begin{bmatrix} & & \\ & & \\ & & \end{bmatrix}$$

Example

$$B = \begin{bmatrix} -1 & 1 & 0 \\ 3 & 2 & -2 \end{bmatrix}$$

$$A = \begin{bmatrix} 1 & 2 \\ 5 & 0 \\ 1 & 3 \end{bmatrix} \quad AB = \begin{bmatrix} 5 & & \\ & & \\ & & \end{bmatrix}$$

$$c_{11} = a_{11}b_{11} + a_{12}b_{21} = 1 \cdot -1 + 2 \cdot 3 = 5$$

Example

$$B = \begin{bmatrix} -1 & 1 & 0 \\ 3 & 2 & -2 \end{bmatrix}$$

$$A = \begin{bmatrix} 1 & 2 \\ 5 & 0 \\ 1 & 3 \end{bmatrix} \quad AB = \begin{bmatrix} 5 & 5 & \\ & & \\ & & \end{bmatrix}$$

$$c_{12} = a_{11}b_{12} + a_{12}b_{22} = 1 \cdot 1 + 2 \cdot 2 = 5$$

Example

$$B = \begin{bmatrix} -1 & 1 & 0 \\ 3 & 2 & -2 \end{bmatrix}$$

$$A = \begin{bmatrix} 1 & 2 \\ 5 & 0 \\ 1 & 3 \end{bmatrix} \quad AB = \begin{bmatrix} 5 & 5 & -4 \\ & & \\ & & \end{bmatrix}$$

$$c_{13} = a_{11}b_{13} + a_{12}b_{23} = 1 \cdot 0 + 2 \cdot -2 = -4$$

Example

$$B = \begin{bmatrix} -1 & 1 & 0 \\ 3 & 2 & -2 \end{bmatrix}$$

$$A = \begin{bmatrix} 1 & 2 \\ 5 & 0 \\ 1 & 3 \end{bmatrix} \quad AB = \begin{bmatrix} 5 & 5 & -4 \\ -5 & & \\ & & \end{bmatrix}$$

$$c_{21} = a_{21}b_{11} + a_{22}b_{21} = 5 \cdot -1 + 0 \cdot 3 = -5$$

Example

$$B = \begin{bmatrix} -1 & 1 & 0 \\ 3 & 2 & -2 \end{bmatrix}$$

$$A = \begin{bmatrix} 1 & 2 \\ 5 & 0 \\ 1 & 3 \end{bmatrix} \quad AB = \begin{bmatrix} 5 & 5 & -4 \\ -5 & 5 & 0 \\ 8 & 7 & -6 \end{bmatrix}$$

Practice

Multiply the matrices A and B.

$$B = \begin{bmatrix} 1 & 0 & -5 \\ 2 & 4 & -6 \end{bmatrix}$$

$$A = \begin{bmatrix} 2 & -4 \\ 3 & 1 \end{bmatrix} \quad AB = \begin{bmatrix} \quad & \quad & \quad \\ \quad & \quad & \quad \end{bmatrix}$$

Practice

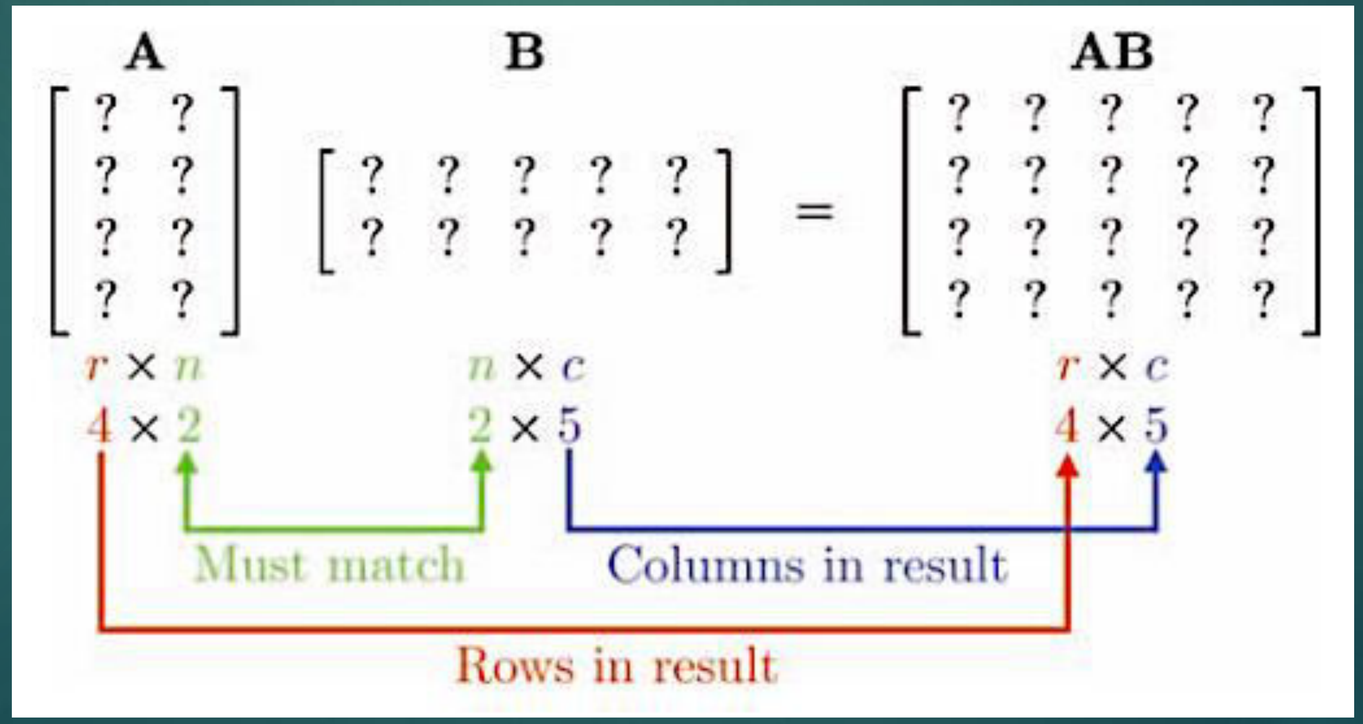
Multiply the matrices A and B.

$$B = \begin{bmatrix} 1 & 0 & -5 \\ 2 & 4 & -6 \end{bmatrix}$$

$$A = \begin{bmatrix} 2 & -4 \\ 3 & 1 \end{bmatrix} \qquad AB = \begin{bmatrix} -6 & -16 & 14 \\ 5 & 4 & -21 \end{bmatrix}$$

Matrix Multiplication Constraint

- Matrices cannot be multiplied unless the number of columns of the first matrix matches the number of rows in the second matrix.



Multiplying by Scalar

$$\mathbf{M} = \begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{bmatrix}$$

$$k\mathbf{M} = k \begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{bmatrix} = \begin{bmatrix} k\ m_{11} & k\ m_{12} & k\ m_{13} \\ k\ m_{21} & k\ m_{22} & k\ m_{23} \\ k\ m_{31} & k\ m_{32} & k\ m_{33} \end{bmatrix}$$

Multiply a matrix by a scalar
multiplies each element accordingly

Multiplying by Vector

$$\mathbf{b} = \mathbf{M}\ \mathbf{a}$$

$$\begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

Multiplying a vector by a matrix is just a compact way of writing a bunch of dot products

Row vs. Column

"row vectors" "column vector"

$$\begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

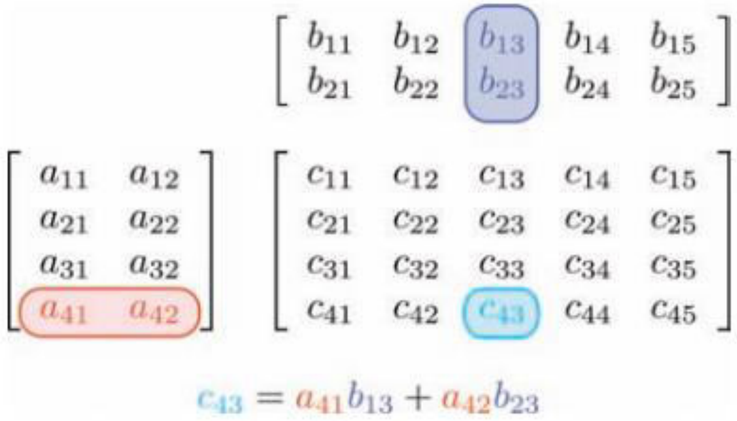
$$\begin{bmatrix} b_1 & b_2 & b_3 \end{bmatrix} = \begin{bmatrix} a_1 & a_2 & a_3 \end{bmatrix} \begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{bmatrix}$$

"row vector"

"column vectors"

Matrix Multiplication (Visually)

$$\mathbf{C} = \mathbf{AB}$$



$$c_{ij} = \mathbf{A}.\mathrm{row}[i] \cdot \mathbf{B}.\mathrm{col}[j]$$

Matrix Multiplication (Convention)

$$\mathbf{C} = \mathbf{AB}$$

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

$$\begin{bmatrix} c_{11} & c_{12} & c_{13} & c_{14} & c_{15} \\ c_{21} & c_{22} & c_{23} & c_{24} & c_{25} \\ c_{31} & c_{32} & c_{33} & c_{34} & c_{35} \\ c_{41} & c_{42} & c_{43} & c_{44} & c_{45} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \\ a_{41} & a_{42} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} & b_{15} \\ b_{21} & b_{22} & b_{23} & b_{24} & b_{25} \end{bmatrix}$$

$$c_{24} = a_{21} b_{14} + a_{22} b_{24}$$

Let’s try it in NumPy…

Coming up...

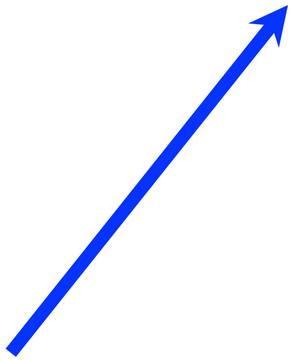
- Transformations

Additional Linear Algebra

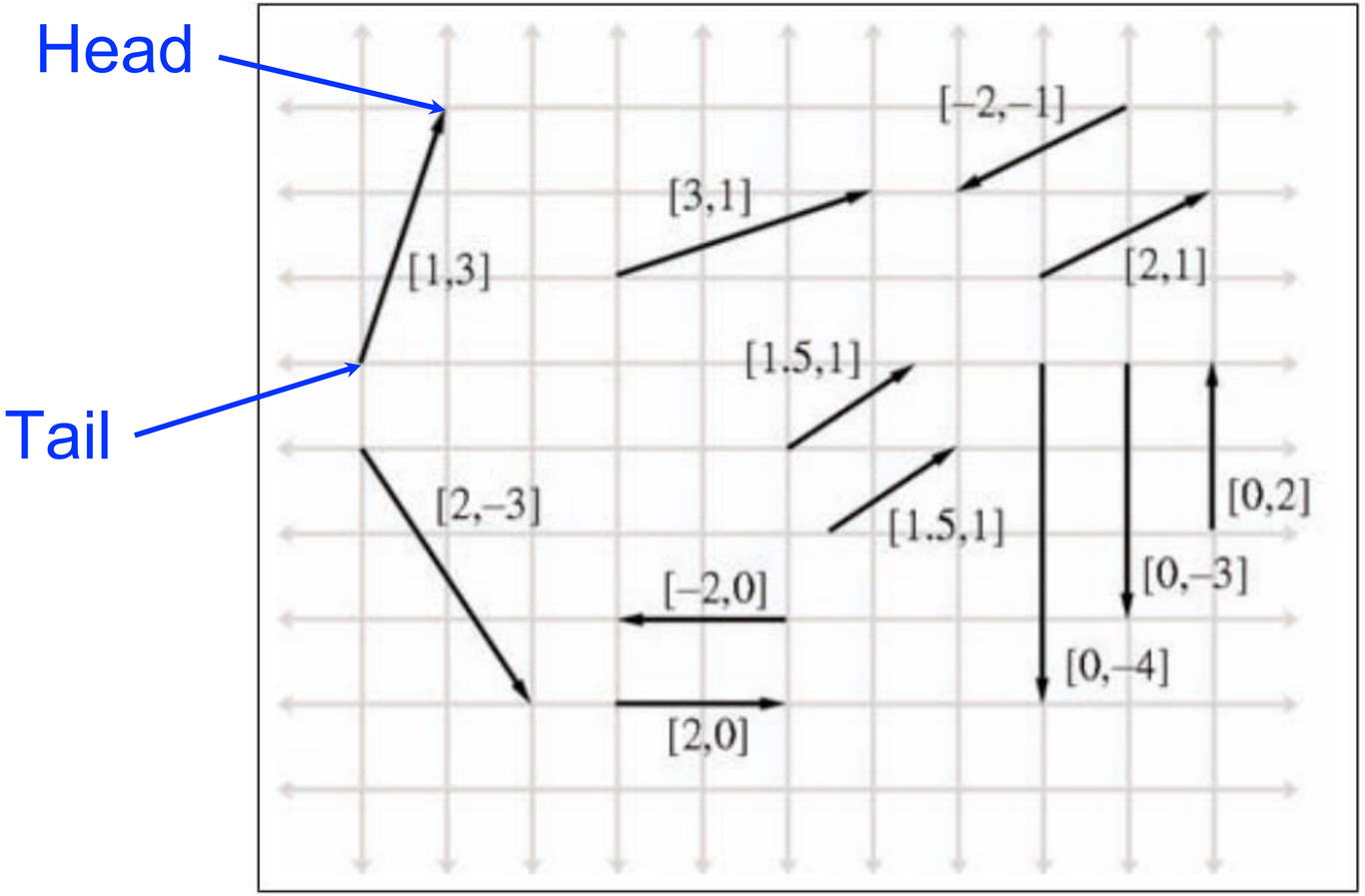
CSCI 3800/6820: Computer Graphics

Vectors

- Direction
- Magnitude (length)



Vectors as Displacements



Zero Vector

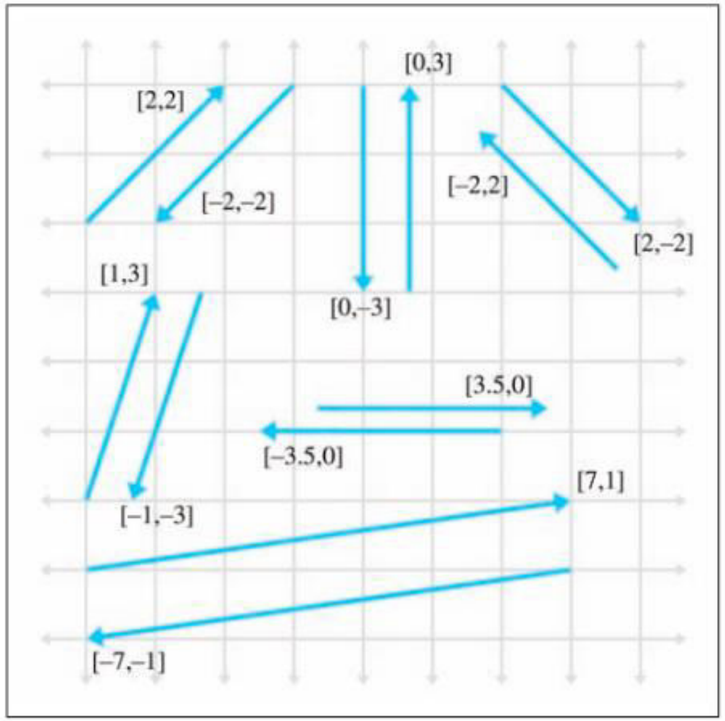
- The *zero vector* is all zeroes
- No displacement
- Magnitude is 0
- Direction is undefined

$$\mathbf{0} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

Negating Vectors

$$\mathbf{v} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

$$-\mathbf{v} = \begin{bmatrix} -x \\ -y \\ -z \end{bmatrix}$$

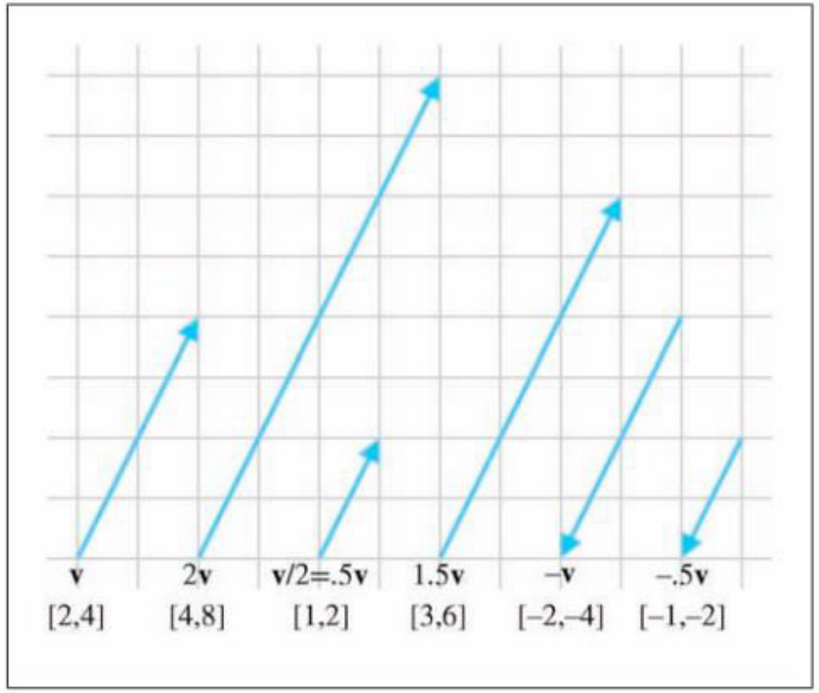


The negative of a vector has the same magnitude in the opposite direction

Scaling Vectors

$$\mathbf{v} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

$$k\ \mathbf{v} = \begin{bmatrix} k\ x \\ k\ y \\ k\ z \end{bmatrix}$$



Multiplying by a constant multiplies each element
-- multiplies magnitude, same (or opposite) direction

Adding Vectors

$$\mathbf{a} = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} \qquad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

$$\mathbf{a} + \mathbf{b} = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} a_1 + b_1 \\ a_2 + b_2 \\ a_3 + b_3 \end{bmatrix}$$

Point-wise add the elements

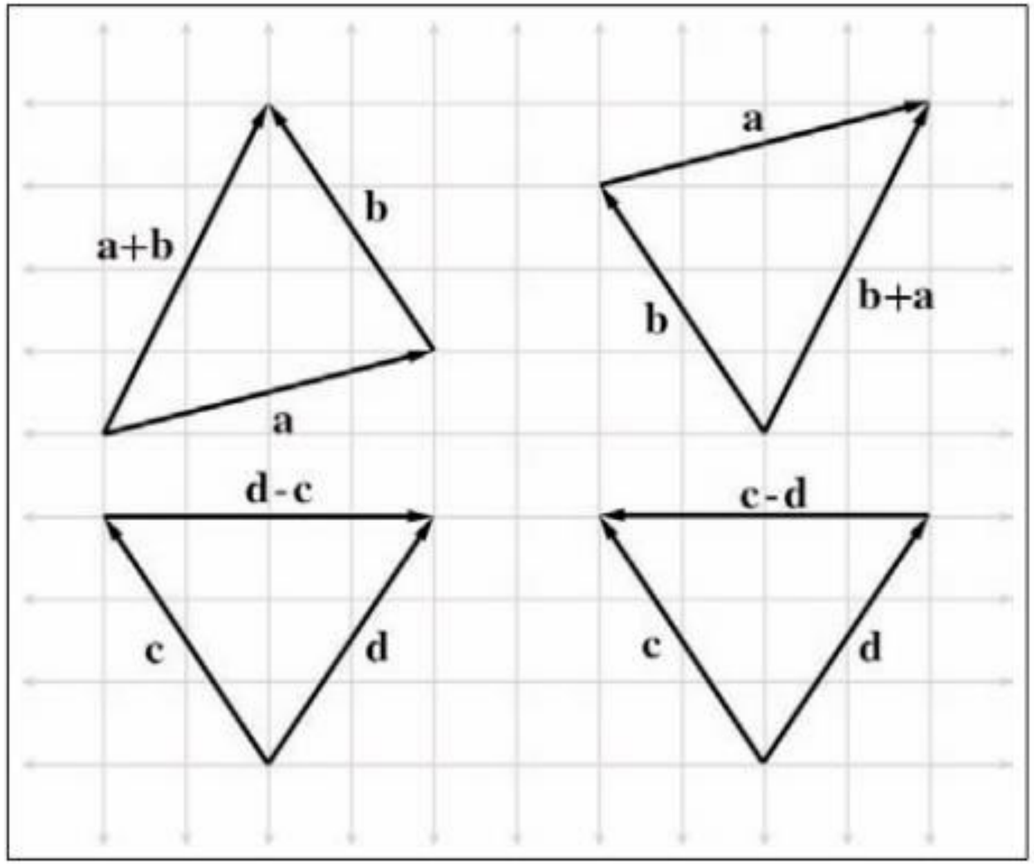
Subtracting Vectors

$$\mathbf{a} = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} \qquad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

$$\mathbf{a} - \mathbf{b} = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} - \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} a_1 - b_1 \\ a_2 - b_2 \\ a_3 - b_3 \end{bmatrix}$$

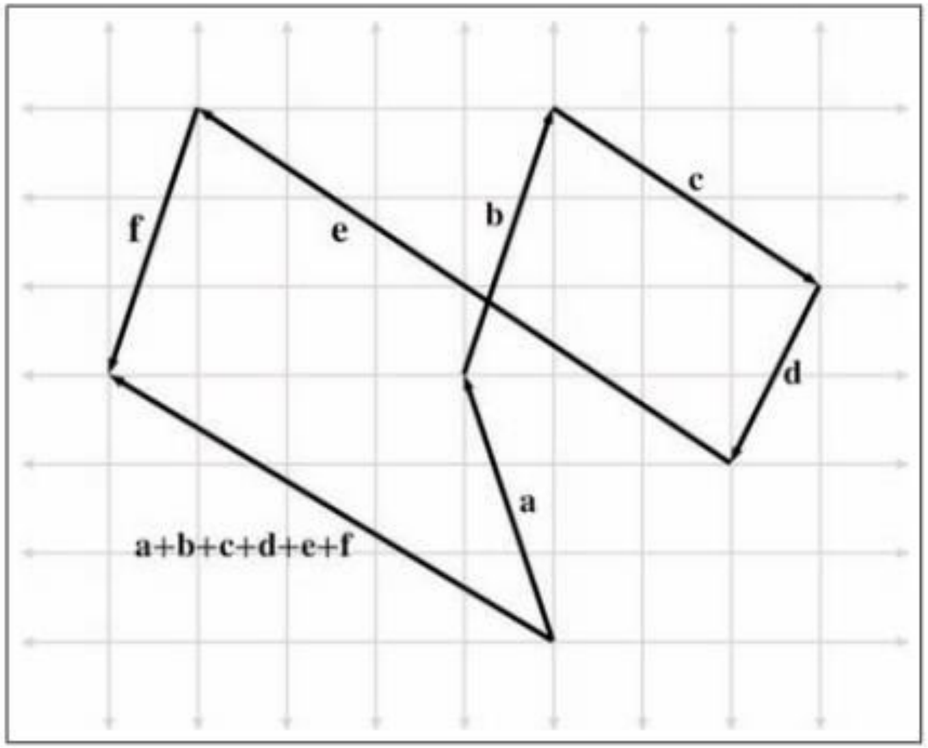
Point-wise subtract the elements

Geometric Interpretation



“Go this way, then go that way”

Geometric Interpretation

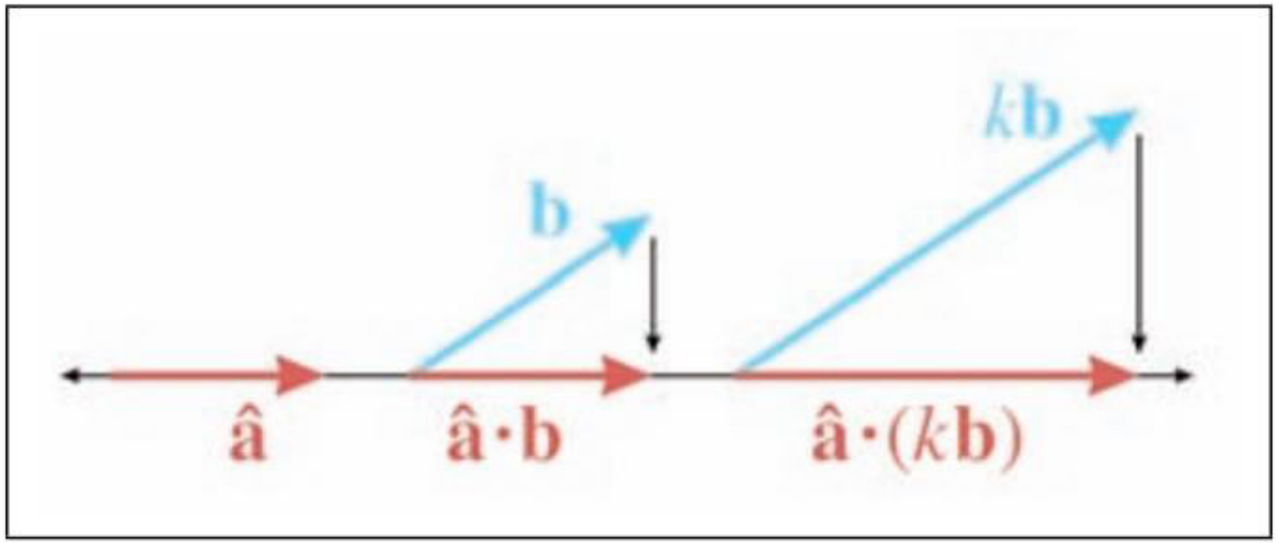


Vector Dot Products

$$\mathbf{a} \cdot \mathbf{b} = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} \cdot \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} = \sum_{i=1}^{n} a_i b_i$$

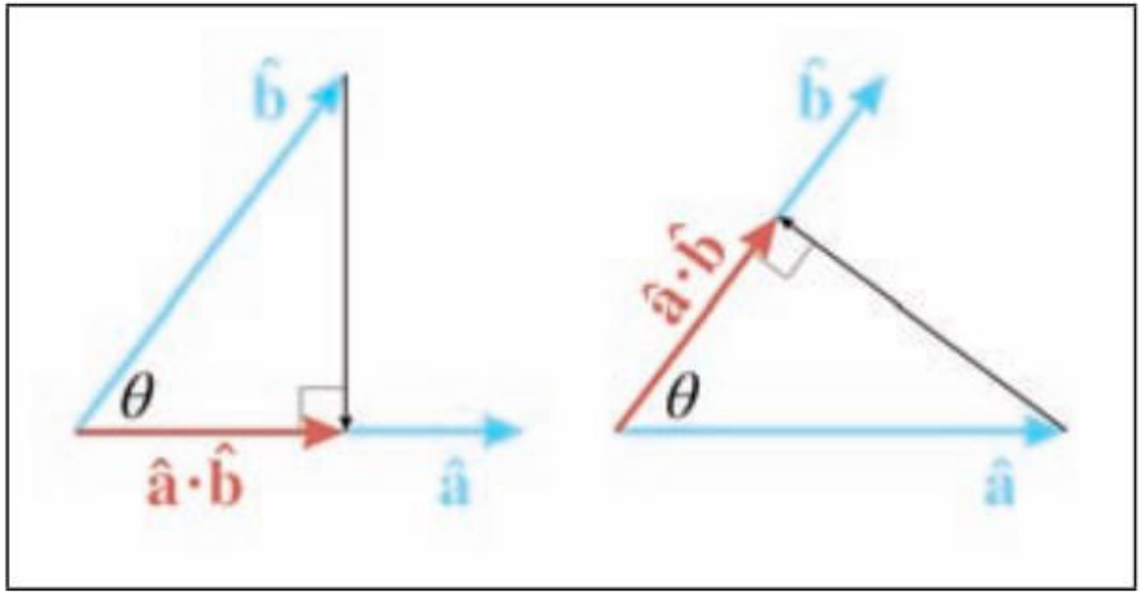
One of the most commonly used
vector operations in graphics

Scalar Multiplication



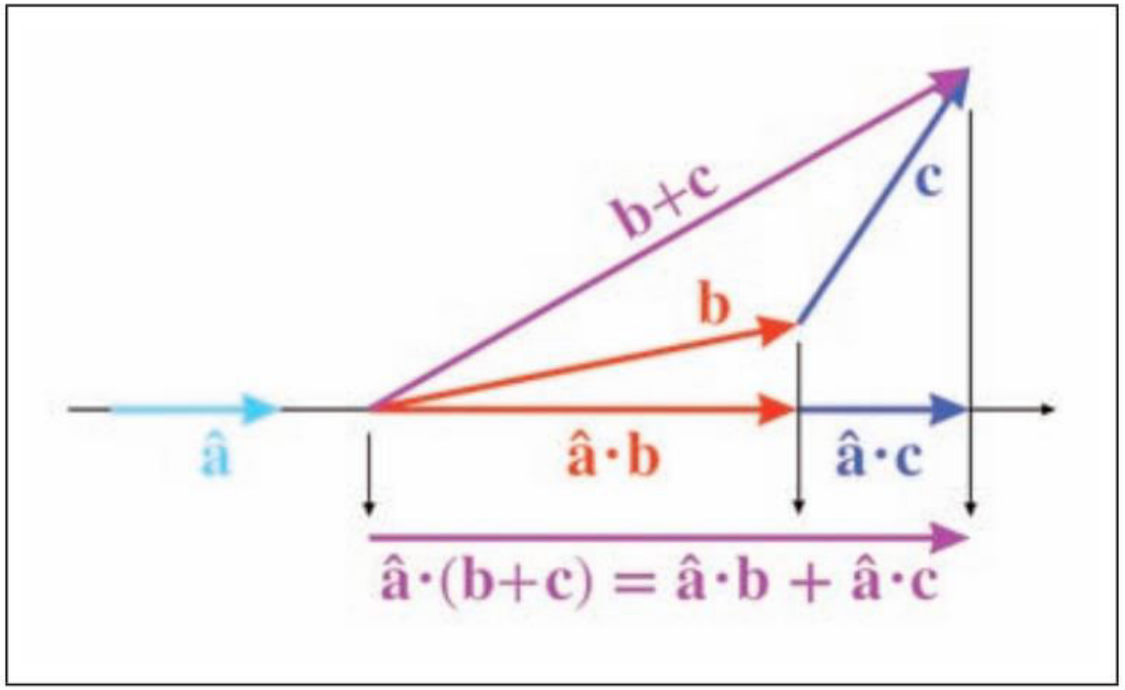
$$\mathbf{a}\cdot(k\mathbf{b}) = k(\mathbf{a}\cdot\mathbf{b}) = (k\mathbf{a})\cdot\mathbf{b}$$

Commutative



$$\mathbf{a}\cdot\mathbf{b} = \mathbf{b}\cdot\mathbf{a}$$

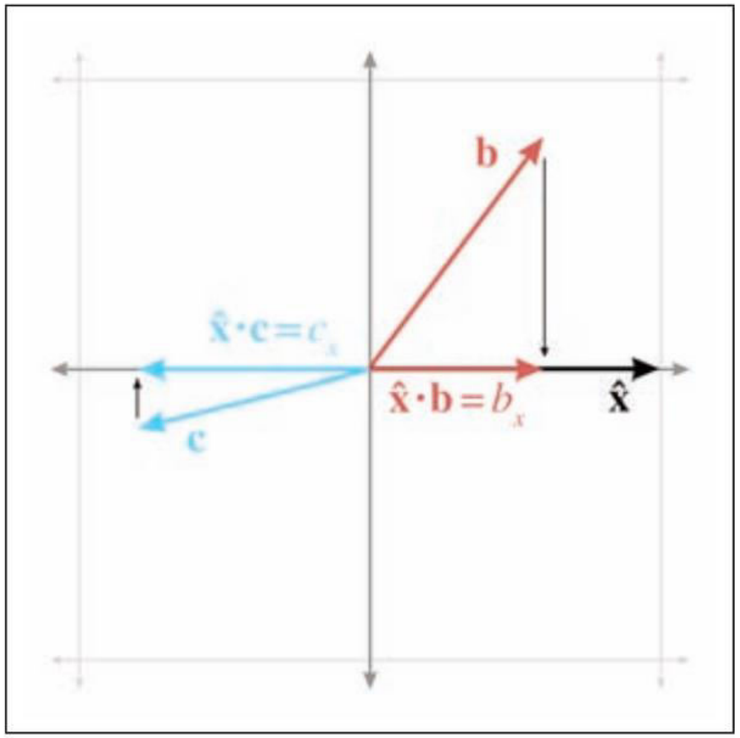
Distributes Over Addition



$$\mathbf{a} \cdot (\mathbf{b} + \mathbf{c}) = \mathbf{a} \cdot \mathbf{b} + \mathbf{a} \cdot \mathbf{c}$$

Coordinates

- To get coordinates, just project onto the axis directions
- Really useful property that we'll come back to



Angles

$$\mathbf{u} = \|\mathbf{u}\| \ \ \hat{\mathbf{u}}$$

$$\mathbf{v} = \|\mathbf{v}\| \ \ \hat{\mathbf{v}}$$

$$\begin{aligned} \mathbf{u} \cdot \mathbf{v} &= \|\mathbf{u}\| \, \|\mathbf{v}\| \, (\hat{\mathbf{u}} \cdot \hat{\mathbf{v}}) \\ &= \|\mathbf{u}\| \, \|\mathbf{v}\| \cos \theta_{uv} \end{aligned}$$

The dot product of two vectors is the product of their lengths times the cosine of the angle between them

Lengths

$$\mathbf{v} \cdot \mathbf{v} = \|\mathbf{v}\|^2$$

The dot product of something with itself
is its own length squared

Tip: lots of “distance” tests only need squared distance

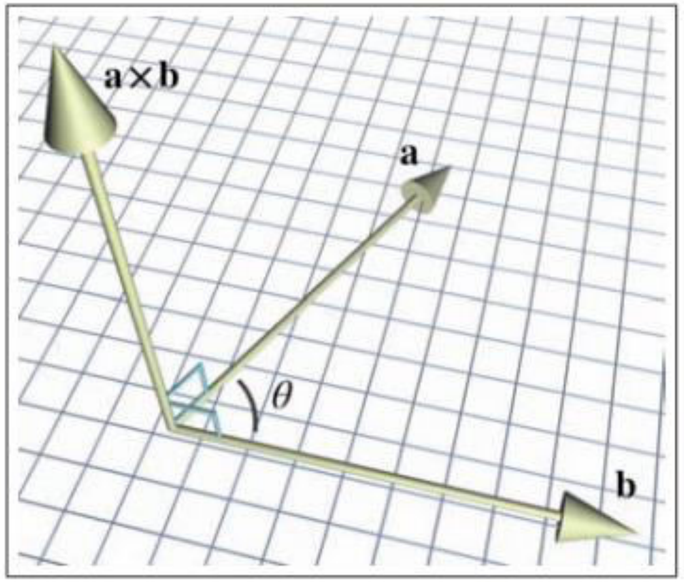
Cross Product

$$\begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} \times \begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix} = \begin{bmatrix} y_1 z_2 - z_1 y_2 \\ z_1 x_2 - x_1 z_2 \\ x_1 y_2 - y_1 x_2 \end{bmatrix}$$

Result is a vector

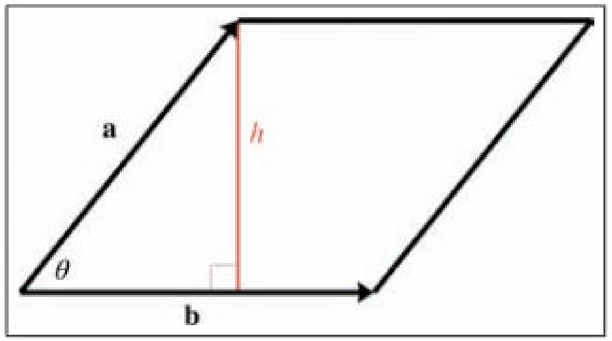
Only done in 3D

Geometric Interpretation



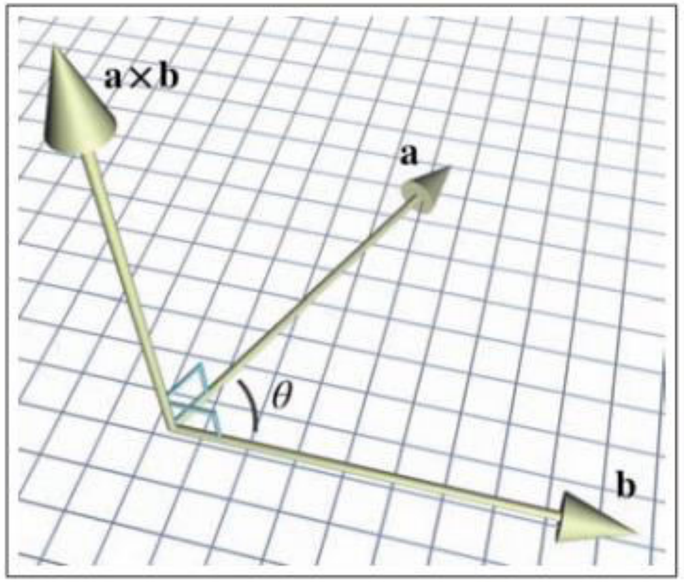
The cross product of two vectors is
another vector orthogonal to the two
(really useful property in 3D geometry!)

Geometric Interpretation



The length of the cross product of two vectors
is the area of the parallelogram spanned by the two

Geometric Interpretation



$$\|\mathbf{a} \times \mathbf{b}\| = \|\mathbf{a}\| \, \|\mathbf{b}\| \sin \theta_{ab}$$

Linear Algebra Identities

Identity	Comments
$\mathbf{a} + \mathbf{b} = \mathbf{b} + \mathbf{a}$	Commutative property of vector addition
$\mathbf{a} - \mathbf{b} = \mathbf{a} + (-\mathbf{b})$	Definition of vector subtraction
$(\mathbf{a} + \mathbf{b}) + \mathbf{c} = \mathbf{a} + (\mathbf{b} + \mathbf{c})$	Associative property of vector addition
$s(t\mathbf{a}) = (st)\mathbf{a}$	Associative property of scalar multiplication
$k(\mathbf{a} + \mathbf{b}) = k\mathbf{a} + k\mathbf{b}$	Scalar multiplication distributes over vector addition
$\lVert k\mathbf{a} \rVert = \lvert k \rvert \lVert \mathbf{a} \rVert$	Multiplying a vector by a scalar scales the magnitude by a factor equal to the absolute value of the scalar
$\lVert \mathbf{a} \rVert \geq 0$	The magnitude of a vector is nonnegative
$\lVert \mathbf{a} \rVert^2 + \lVert \mathbf{b} \rVert^2 = \lVert \mathbf{a} + \mathbf{b} \rVert^2$	The Pythagorean theorem applied to vector addition.
$\lVert \mathbf{a} \rVert + \lVert \mathbf{b} \rVert \geq \lVert \mathbf{a} + \mathbf{b} \rVert$	Triangle rule of vector addition. (No side can be longer than the sum of the other two sides.)
$\mathbf{a} \cdot \mathbf{b} = \mathbf{b} \cdot \mathbf{a}$	Commutative property of dot product
$\lVert \mathbf{a} \rVert = \sqrt{\mathbf{a} \cdot \mathbf{a}}$	Vector magnitude defined using dot product
$k(\mathbf{a} \cdot \mathbf{b}) = (k\mathbf{a}) \cdot \mathbf{b} = \mathbf{a} \cdot (k\mathbf{b})$	Associative property of scalar multiplication with dot product
$\mathbf{a} \cdot (\mathbf{b} + \mathbf{c}) = \mathbf{a} \cdot \mathbf{b} + \mathbf{a} \cdot \mathbf{c}$	Dot product distributes over vector addition and subtraction
$\mathbf{a} \times \mathbf{a} = \mathbf{0}$	The cross product of any vector with itself is the zero vector. (Because any vector is parallel with itself.)
$\mathbf{a} \times \mathbf{b} = -(\mathbf{b} \times \mathbf{a})$	Cross product is anticommutative.
$\mathbf{a} \times \mathbf{b} = (-\mathbf{a}) \times (-\mathbf{b})$	Negating both operands to the cross product results in the same vector.
$k(\mathbf{a} \times \mathbf{b}) = (k\mathbf{a}) \times \mathbf{b} = \mathbf{a} \times (k\mathbf{b})$	Associative property of scalar multiplication with cross product.
$\mathbf{a} \times (\mathbf{b} + \mathbf{c}) = \mathbf{a} \times \mathbf{b} + \mathbf{a} \times \mathbf{c}$	Cross product distributes over vector addition and subtraction.

A little more on matrices

Transposing

$$\mathbf{M} = \begin{bmatrix} 3 & 1 & 8 & 5 \\ -1 & 4 & -3 & 3 \\ 2 & 0 & -1 & 4 \end{bmatrix} \qquad \mathbf{M}^T = \begin{bmatrix} 3 & -1 & 2 \\ 1 & 4 & 0 \\ 8 & -3 & -1 \\ 5 & 3 & 4 \end{bmatrix}$$

"M transpose"

The transposition of a matrix simply swaps the rows for the columns

$$\mathbf{M}^T_{ij} = \mathbf{M}_{ji}$$

Determinant

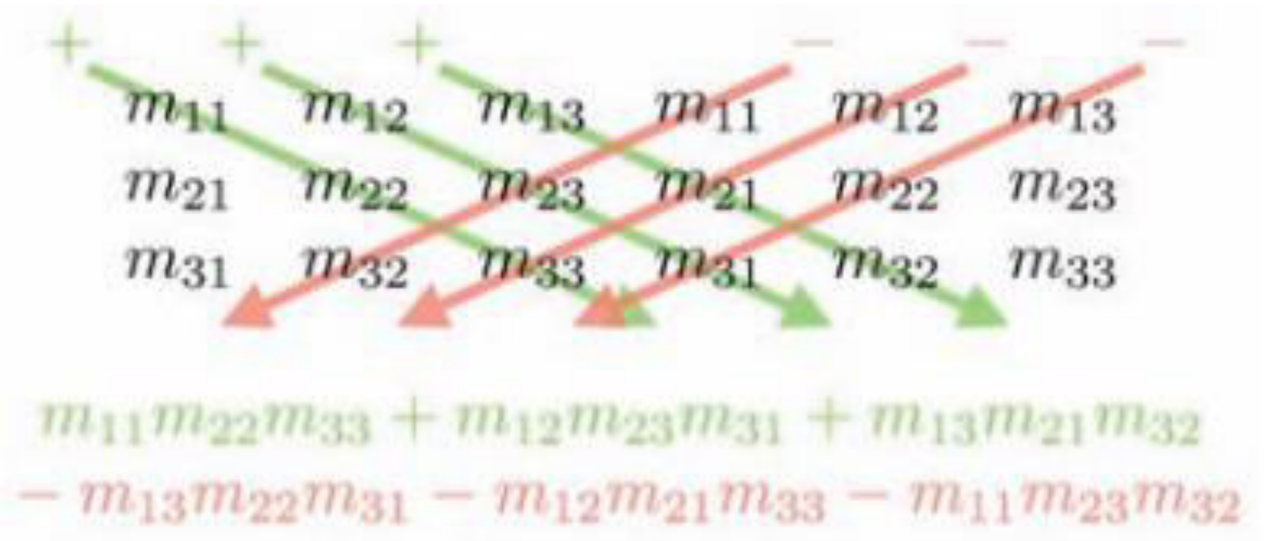
$$|\mathbf{M}| = \begin{vmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{vmatrix} = m_{11}m_{22} - m_{12}m_{21}$$

Determinant

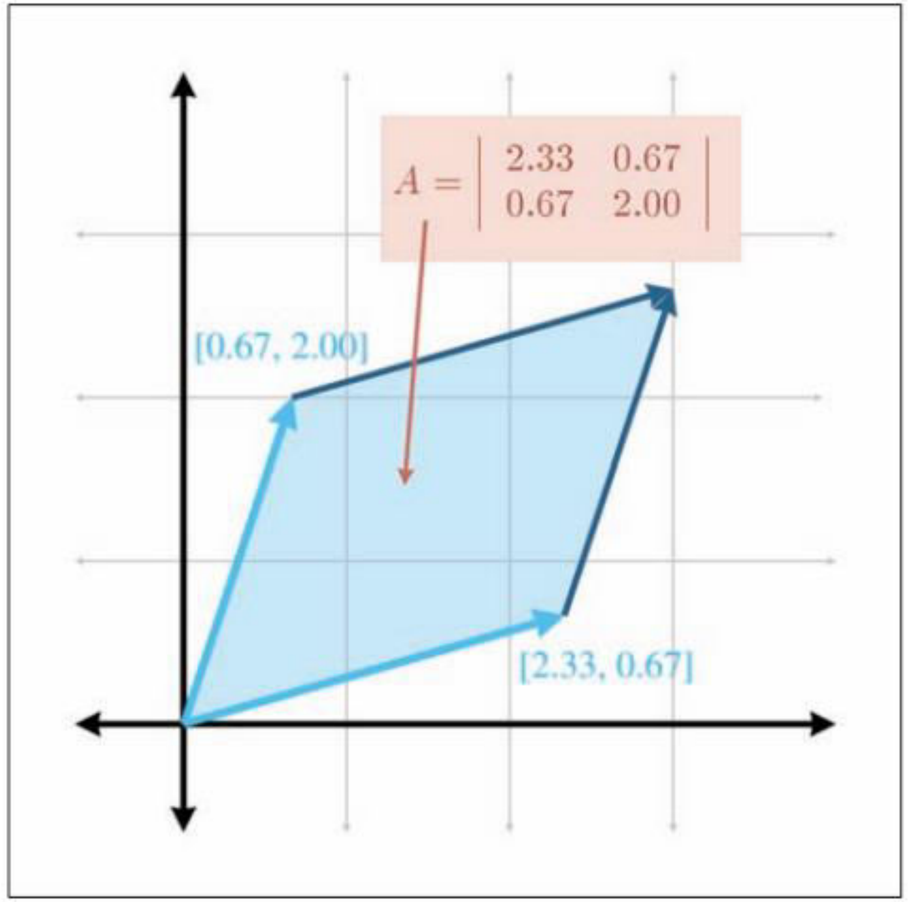
$$|\mathbf{M}| = \begin{vmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{vmatrix} = m_{11}m_{22} - m_{12}m_{21}$$

Determinant

$$|\mathbf{M}| = \begin{vmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{vmatrix}$$



Geometric Interpretation



Properties of Determinants

$$|\mathbf{I}| = 1$$

$$|\mathbf{AB}| = |\mathbf{A}|\,|\mathbf{B}|$$

$$\left|\mathbf{M}^T\right| = |\mathbf{M}|$$

$$\left|\mathbf{M}^{-1}\right| = \frac{1}{|\mathbf{M}|}$$

Linear Independence

A set of vectors is said to be *linearly dependent* if at least one of them can be expressed as a linear combination (weighted sum) of the others:

$$\mathbf{v}_j = \sum_{i \neq j} w_i \ \mathbf{v}_i$$

If not linearly *dependent*, then linearly *independent*

Singular Matrices

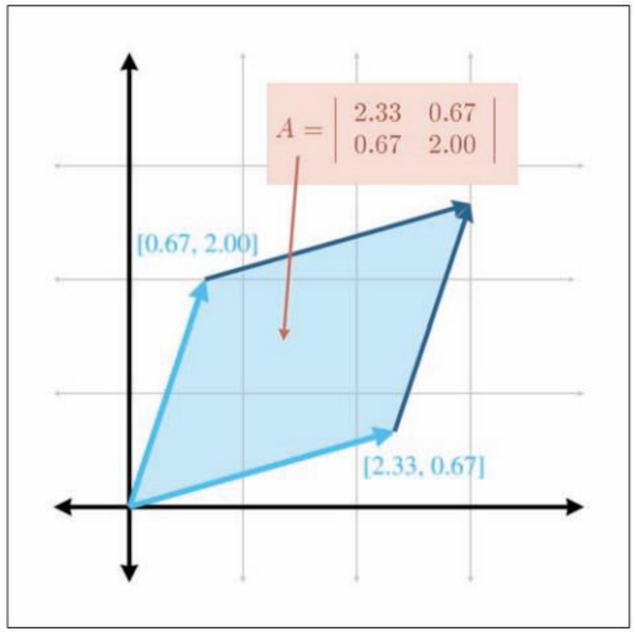
$$\left|\mathbf{M}^{-1}\right| = \frac{1}{|\mathbf{M}|}$$

But what if $|\mathbf{M}| = 0$?

A matrix whose determinant is zero
has no inverse and is said to be *singular*

Singular Matrices

- What does a singular matrix mean geometrically?
- *The rows are linearly dependent*
- “Flattens” the space



Matrix Rank

- The *rank* of a matrix is the number of linearly independent rows
- When used as transforms, matrices with *full rank* transform to the full space
- Singular matrices have *insufficient rank* and collapse to a corresponding subspace
- Geometric interpretation: the rank of a matrix is the dimensionality of the (sub)space that matrix maps to

Orthogonal Matrices

- Two (square) matrices are said to be orthogonal iff

$$\mathbf{M}\mathbf{M}^T = \mathbf{I}$$

- Implies rows are orthonormal vectors

Orthogonal Matrices

- Orthogonal matrices are also easily invertible:

$$\mathbf{M}^{-1} = \mathbf{M}^{T}$$

- Implies

$$|\mathbf{M}| = \left|\mathbf{M}^{-1}\right| = 1$$

Orthogonal Matrices

- All rotation matrices are orthogonal

AND

- All orthogonal matrices are rotations!

Identity Matrix

$$\mathbf{I} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{MI} = \mathbf{IM} = \mathbf{M}$$

Matrix Inversion

The inverse of a matrix is the matrix such that

$$\mathbf{M}\mathbf{M}^{-1} = \mathbf{M}^{-1}\mathbf{M} = \mathbf{I}$$

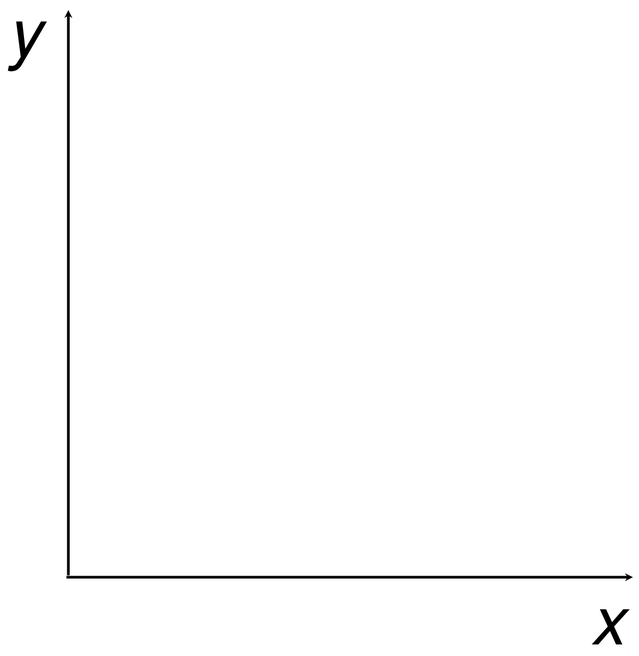
inverse

There are multiple ways to compute the inverse of a matrix, but we won't cover that here

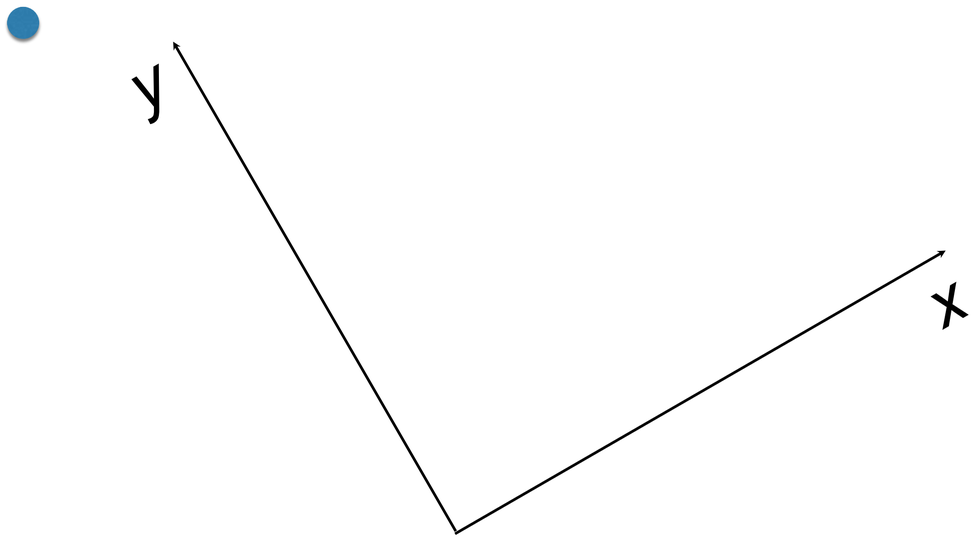
Transformations

CSCI 3800/6820: Computer Graphics

Changes of Coordinates



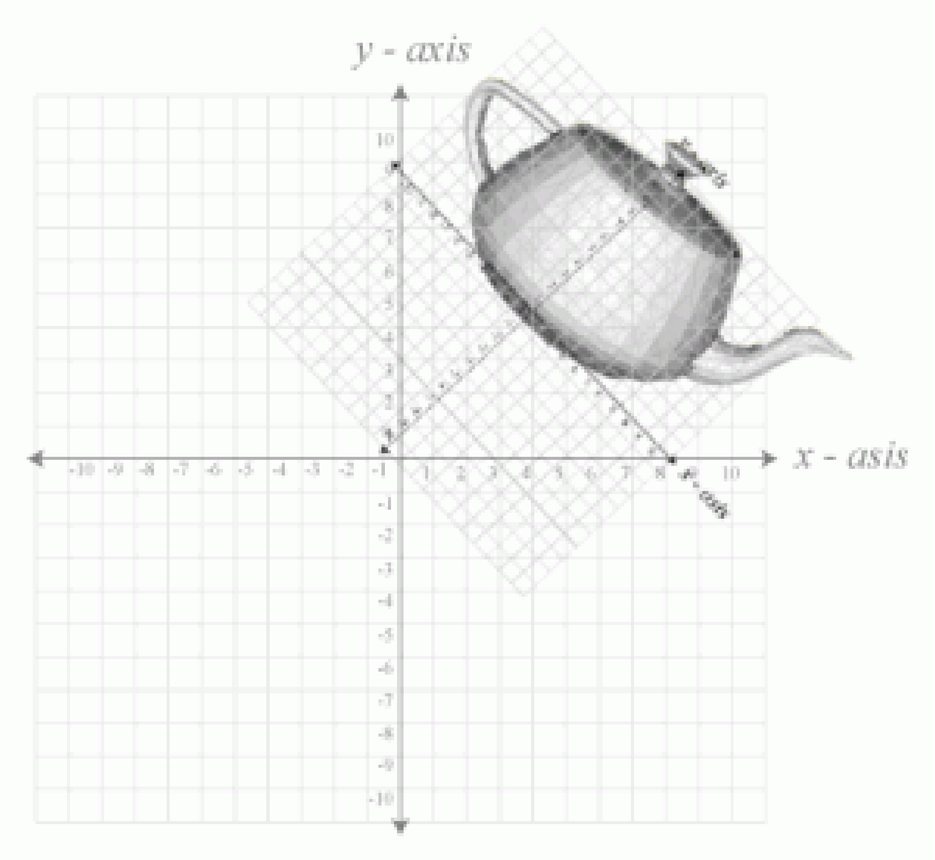
What if we have a point described on one coordinate system…



But we want to describe it in another?

Why Change Coordinates?

- Moving stuff around
- Model in one space, place in another
- Modeling hierarchically
- Where are 3D points *relative to the camera*?
- Geometric tests (Ray Tracing)
- Data transformations



Translation

- *Translating* a coordinate system simply moves the origin
- Or can be thought of as moving the point
- Keeps the x and y directions the same
- Just add desired x, y offsets

$$(x', y') = (x + t_x, y + t_y)$$

OR

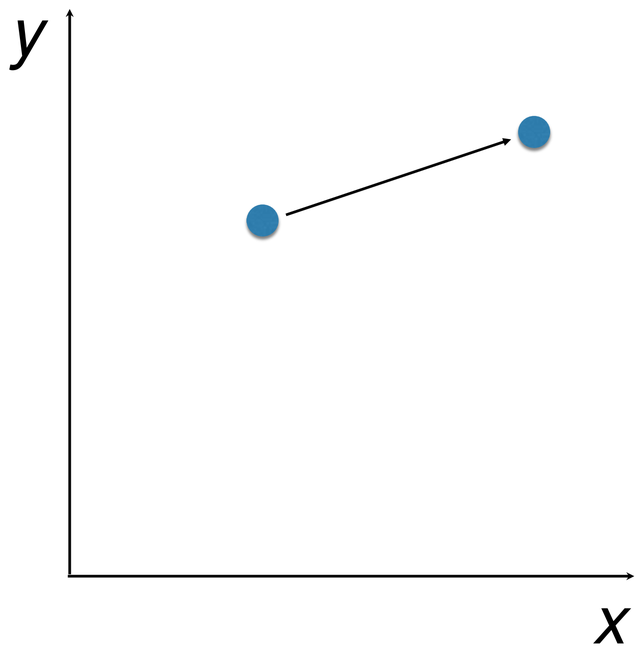
$$\mathbf{p}' = \mathbf{p} + \mathbf{t}$$

$$\mathbf{t} = \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

Example: Translation

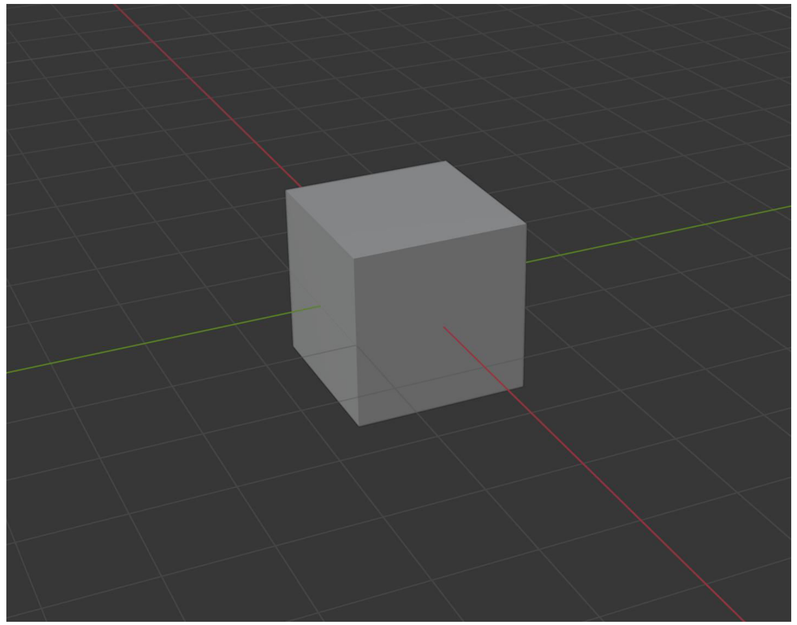
- Suppose that you have a point ***p*** at (100,200) and you want to translate it by [160,50]?

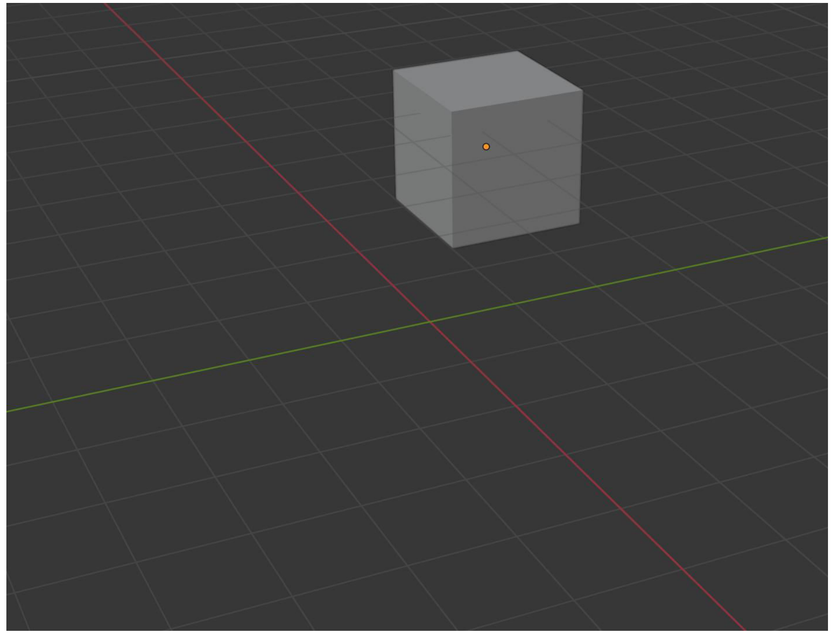
$$
\begin{aligned}
\mathbf{p}' &= \mathbf{p} + \mathbf{t} \\
&= (100, 200) + [160, 50] \\
&= (260, 250)
\end{aligned}
$$



Translation in 3D

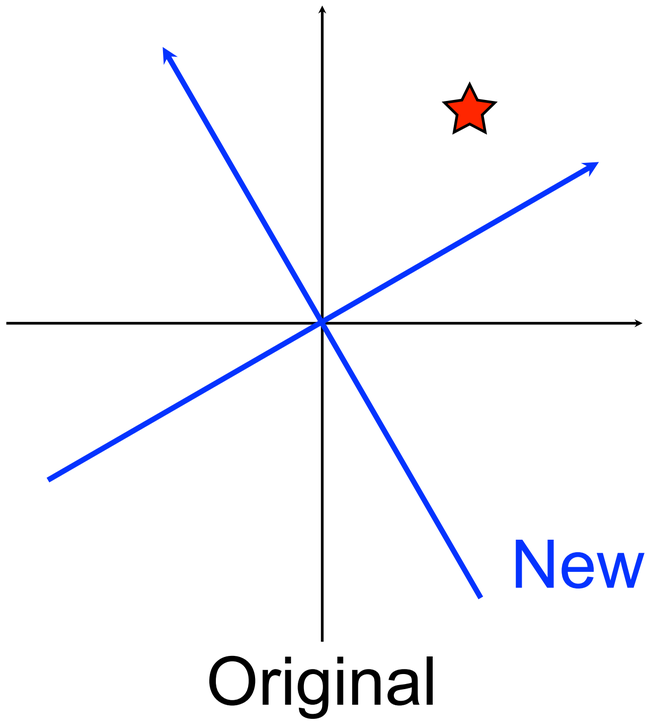
$$\mathrm{t} = \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}$$



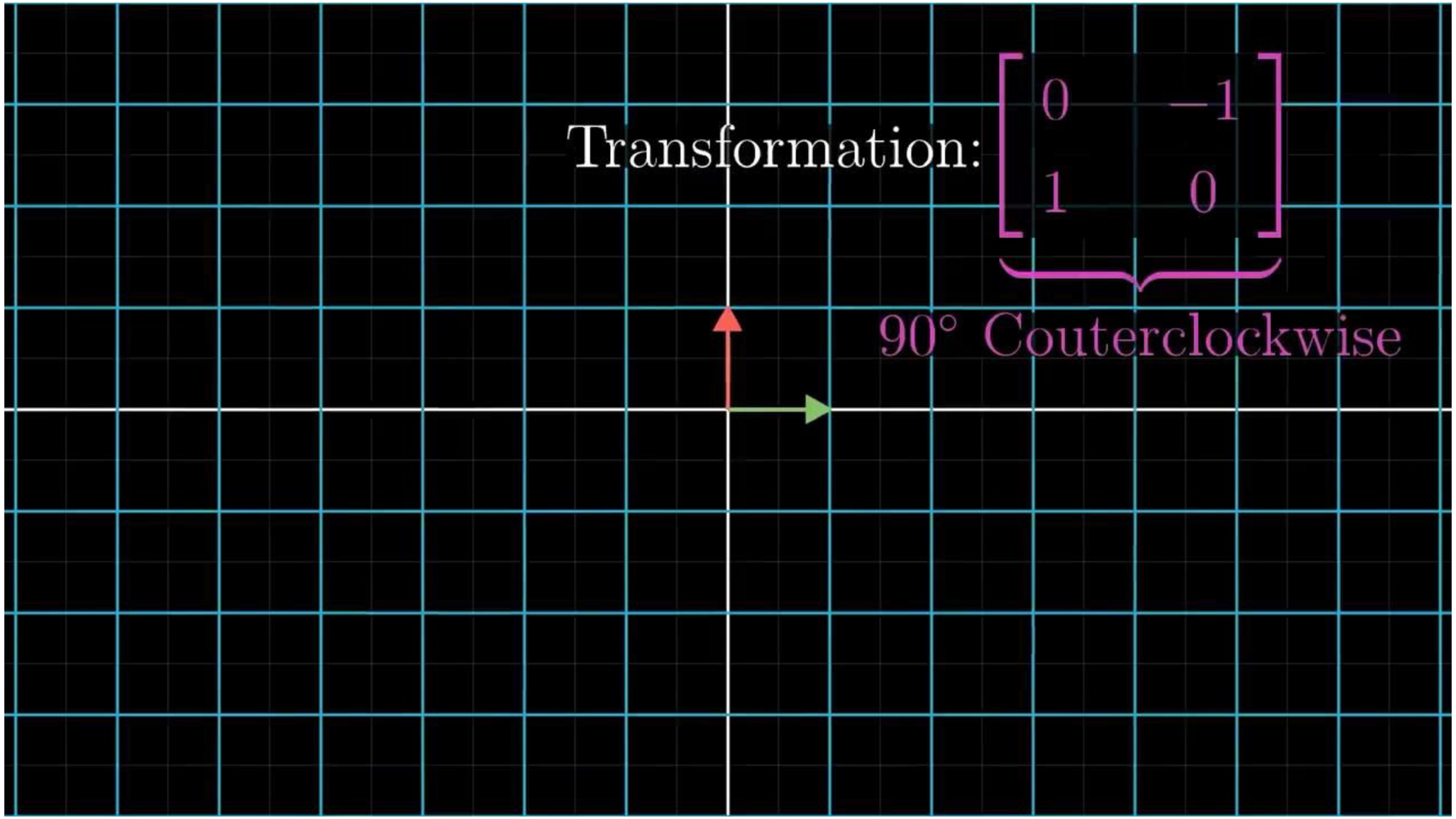


Rotation

- Rotating a coordinate system keeps the origin, turns the axis directions
- Conceptually,
  - The point stays the same and the axes change
  - The axes stay the same and the point rotates around the origin



3Blue1Brown - Rotation

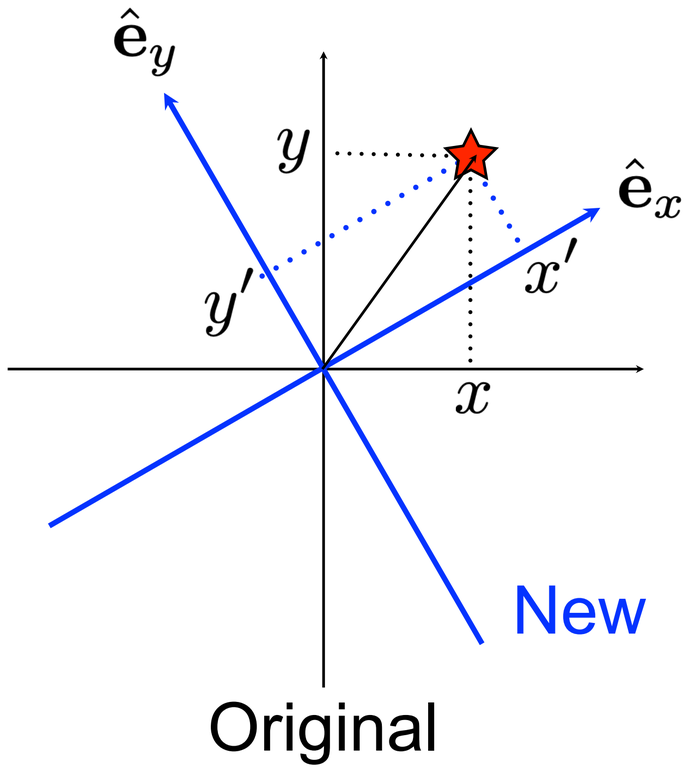


Computing Rotation

- To compute coordinates in the rotated system, just project to each of the new axis directions
- Use dot products!

$$p'_x = \mathbf{p} \cdot \hat{\mathbf{e}}_x$$

$$p'_y = \mathbf{p} \cdot \hat{\mathbf{e}}_y$$

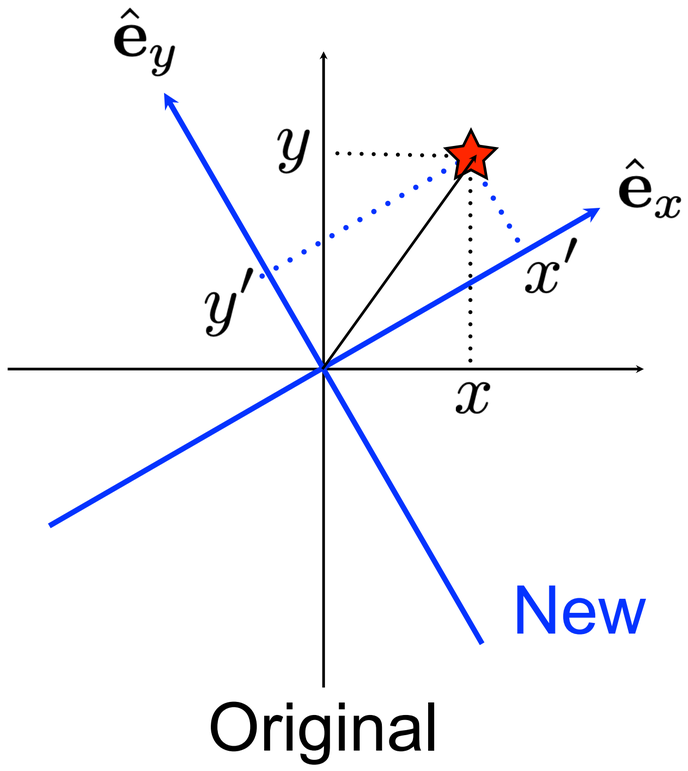


Computing Rotation

- To compute coordinates in the rotated system, just project to each of the new axis directions
- Use dot products!

$$\mathbf{p}' = \begin{bmatrix} e_{x1} & e_{x2} \\ e_{y1} & e_{y2} \end{bmatrix} \mathbf{p}$$

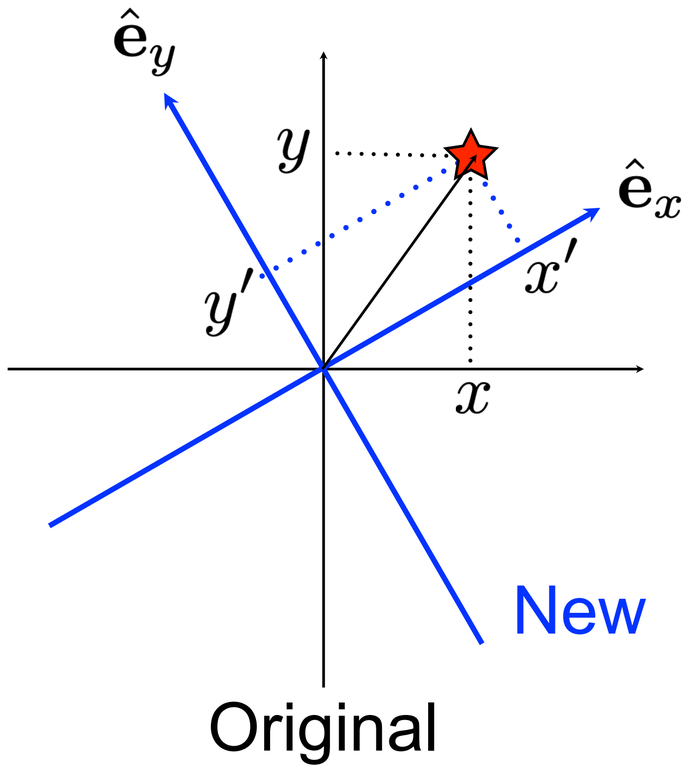
But do it with a matrix!!



Change of Coordinates

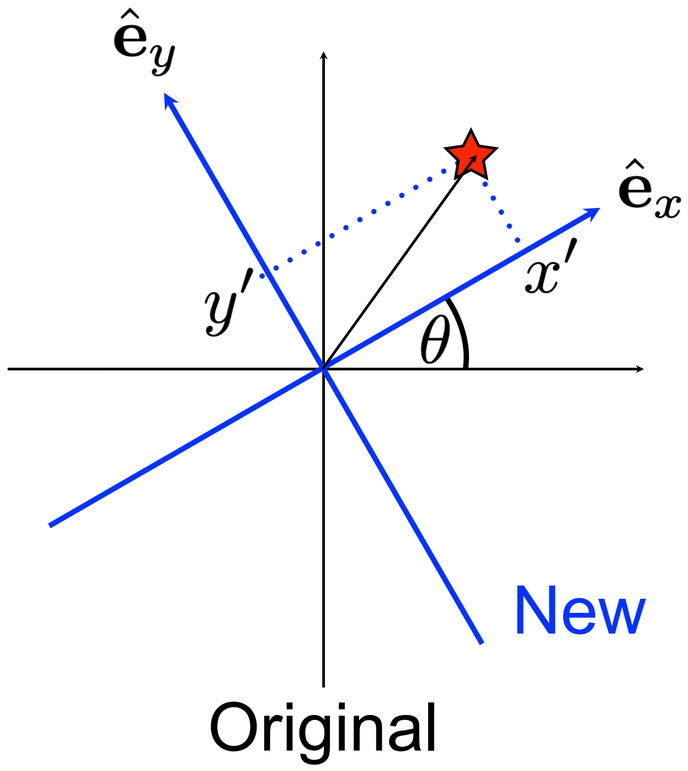
- To compute coordinates in the rotated system, just project to each of the new axis directions
- Use a matrix to do the dot products

$$\mathbf{p}' = \begin{bmatrix} e_{x1} & e_{x2} \\ e_{y1} & e_{y2} \end{bmatrix} \mathbf{p}$$



Rotation

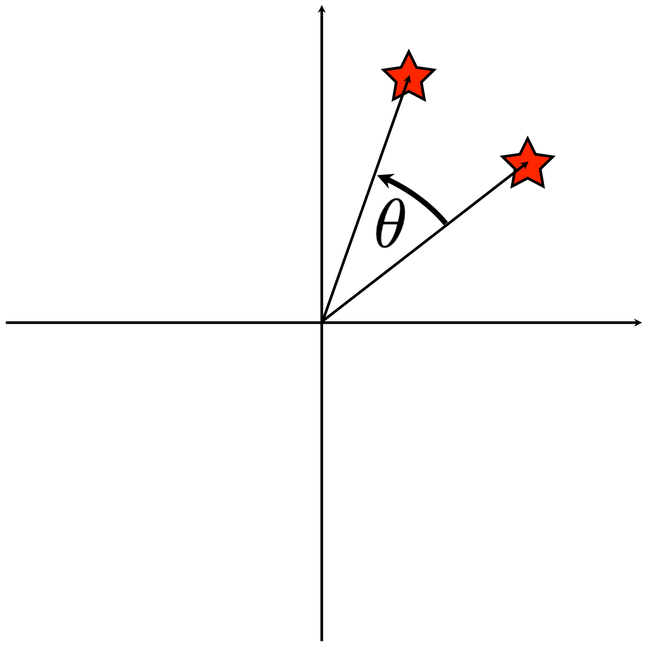
- Rotating the coordinate system one way rotates the point/object the other way
- From here on we'll talk in terms of rotating the point or object



Rotation

$$\mathbf{R}(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

$$\mathbf{R}^{-1}(\theta) = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}$$

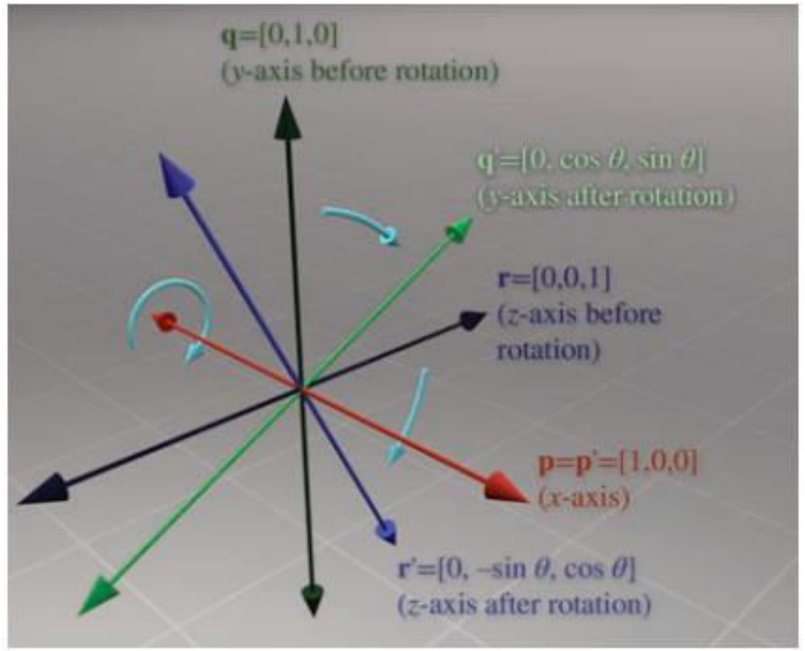


Rotation in 3D

$$\mathbf{R}_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}$$

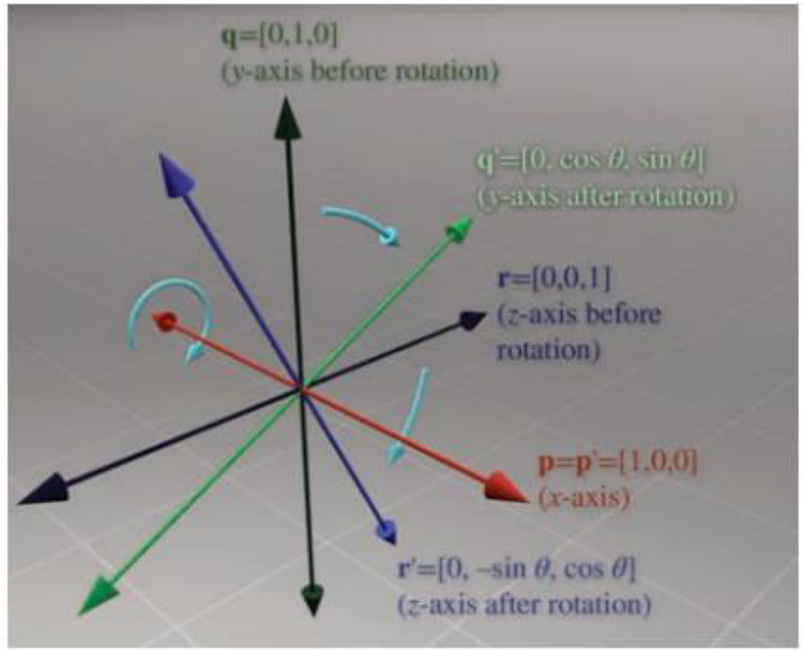
$$\mathbf{R}_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}$$

$$\mathbf{R}_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



Rotation in 3D

$$R = \begin{bmatrix} e_{11} & e_{12} & e_{13} \\ e_{21} & e_{22} & e_{23} \\ e_{31} & e_{32} & e_{33} \end{bmatrix} = R_z R_y R_x$$

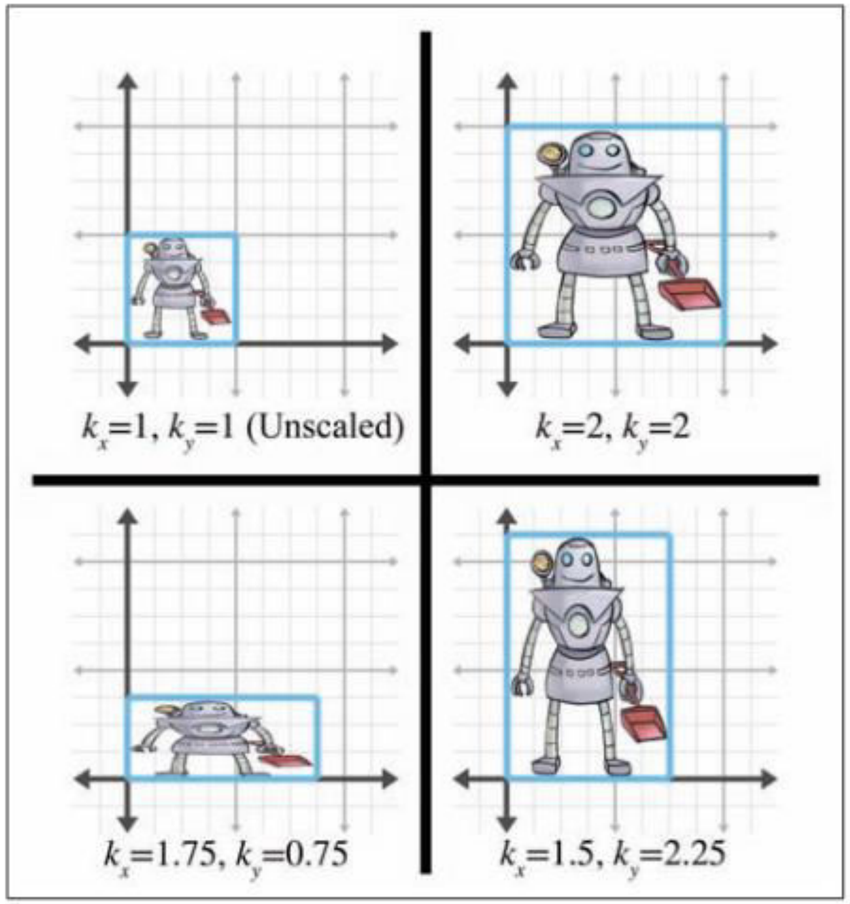


Any arbitrary rotation can be represented by applying axis rotations in succession

Scaling

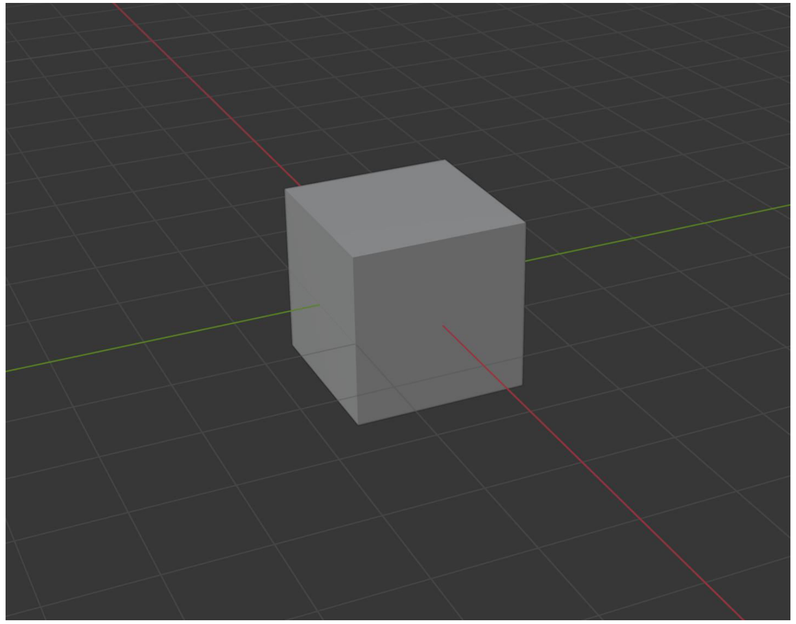
$$\begin{bmatrix} s & 0 \\ 0 & s \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} sx \\ sy \end{bmatrix}$$

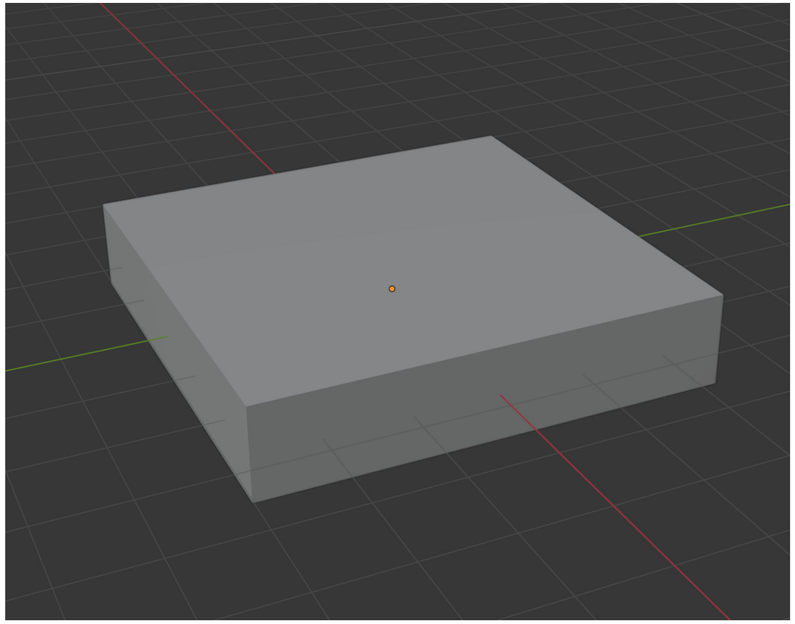
$$\begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} s_x\ x \\ s_y\ y \end{bmatrix}$$



Scaling in 3D

$$S = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{bmatrix}$$





Let’s try it in NumPy…

Composition

$$\mathbf{M}_2 = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \qquad \mathbf{M}_1 = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

$$\mathbf{M}_{12} = \underline{\mathbf{M}_2\mathbf{M}_1}$$

$$\mathbf{M}_{12}\mathbf{p} = (\mathbf{M}_2\mathbf{M}_1)\mathbf{p} = \mathbf{M}_2(\mathbf{M}_1\mathbf{p})$$

But...

- Composition of transformations (matrix multiplication) is really useful
- But translation must be done as a separate operation
- *Can you include translation as a linear transformation*?

Homogeneous Coordinates

- Can include translation as a linear transformation using *homogeneous coordinates*
- Also useful for projection (covered next time)

$$\mathrm{p} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Add additional “homogeneous” element

Homogeneous Matrices

- To make a matrix work with homogeneous coordinates, add an extra row and column
- To do nothing else, make these [0 0 1]

$$\begin{bmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{bmatrix}$$

$$\Downarrow$$

$$\begin{bmatrix} m_{11} & m_{12} & 0 \\ m_{21} & m_{22} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Translation

$$\begin{bmatrix} 1 & 0 & 0 & \boxed{t_x} \\ 0 & 1 & 0 & \boxed{t_y} \\ 0 & 0 & 1 & \boxed{t_z} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x + \boxed{t_x} \\ y + \boxed{t_y} \\ z + \boxed{t_z} \\ 1 \end{bmatrix}$$

(General note: for row vectors, transpose this)

Homogenous Form of 3D Transformations

$$\mathbf{S} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \mathbf{T} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{R} = \begin{bmatrix} e_{11} & e_{12} & e_{13} & 0 \\ e_{21} & e_{22} & e_{23} & 0 \\ e_{31} & e_{32} & e_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Vital for Lab 3!

Let’s try it in NumPy…

Points vs. Vectors

$$\mathrm{p} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \qquad \mathrm{v} = \begin{bmatrix} x \\ y \\ z \\ 0 \end{bmatrix}$$

allows translation

ignores translation

Remember: points have a position and *can* be translated,
but vectors are only a displacement and *cannot* be translated

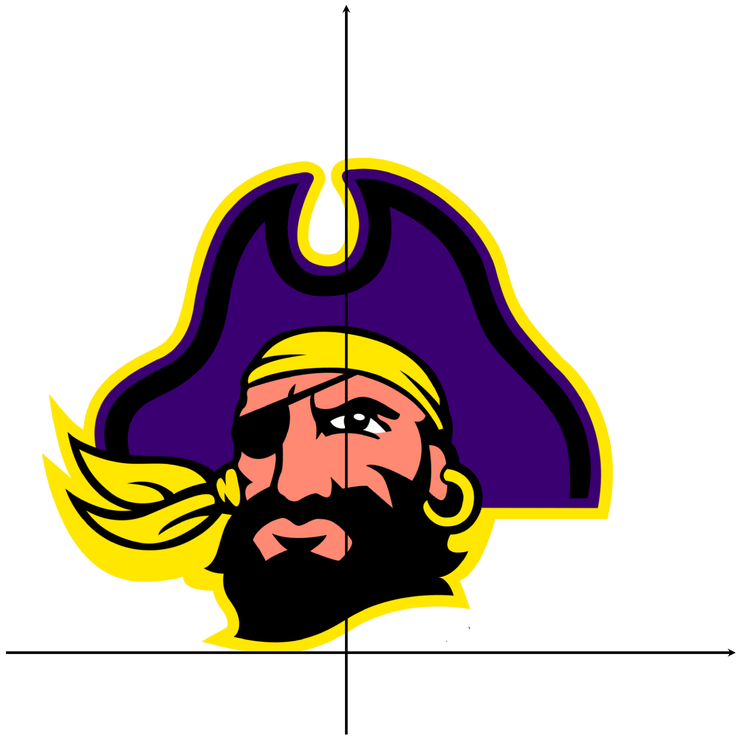
So… why are we doing this?

Spaces

- 3D objects have multiple sets of (x,y,z) coordinates depending on whether you are referencing
  - The Object
  - The Scene
  - The Camera
- These different coordinate systems are called “spaces” and we can use 3D transformations to move from one space to another.

Object Space

- The coordinate system used to define an object
- Choice of origin and coordinate axes are arbitrary, but usually chosen to make object definition the simplest
- Usually centered about (0,0,0).

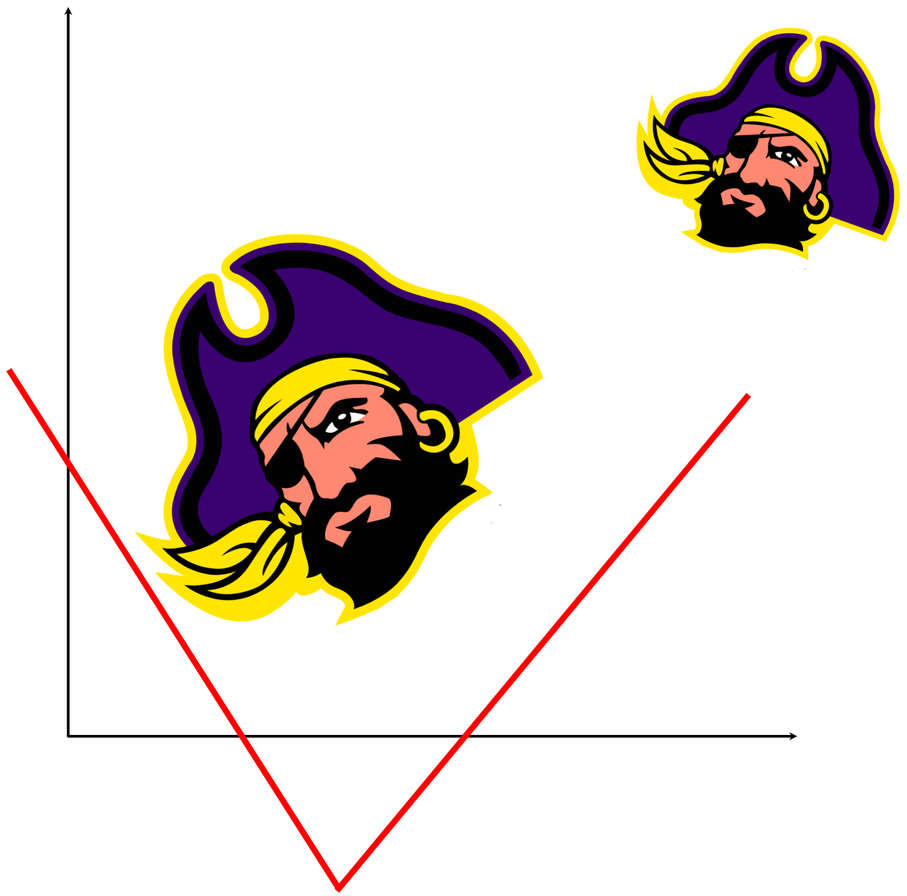


World Space

- The “world space” defines the space in which objects can live in the scene
- Determine the object’s
  - Scale
  - Rotation
  - Translation
- Fixed origin and axes for all objects

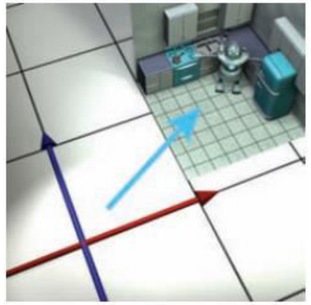
Camera Space

- Where is the object relative to the camera?
- Affected by the camera's
  - Location (translation)
  - Rotation



Object to World

- An object has a *position*, a *size*, and an *orientation*
  - First: **scale** in object space if needed (easiest if the coordinate axes are the natural directions for scaling)
  - Second: **rotate** in object space to desired world-space orientation
  - Third: **translate** (move) to the position in world space



Order matters!

Object to World

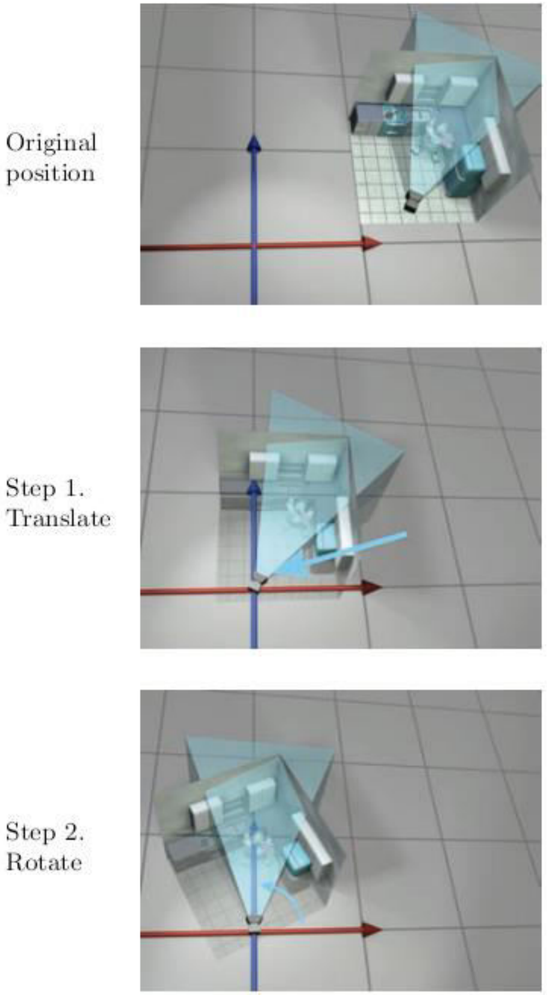
$$p_w = TRSp$$

$$\begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} e_{11} & e_{12} & e_{13} & 0 \\ e_{21} & e_{22} & e_{23} & 0 \\ e_{31} & e_{32} & e_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{\text{Object-to-World Transformation}} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Remember to read from right to left

World to Camera

- Two steps:
  - **Translate**
    everything to be relative to the camera position
  - **Rotate**
    into the camera's viewing orientation



World to Camera

- Suppose that you know
  - Position of camera in world coordinates

$$\mathbf{c} = (c_x, c_y, c_z)$$

  - Orientation of camera as given by
    a set of basic vectors in world coordinates

$$\{\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3\}$$

Camera's x Camera's y Camera's z

World to Camera

- Two steps:
  - **Translate**
    everything to be relative to the camera position
  - **Rotate**
    into the camera's viewing orientation

$$\begin{bmatrix} 1 & 0 & 0 & -c_x \\ 0 & 1 & 0 & -c_y \\ 0 & 0 & 1 & -c_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} e_{11} & e_{12} & e_{13} & 0 \\ e_{21} & e_{22} & e_{23} & 0 \\ e_{31} & e_{32} & e_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Camera's x
Camera's y
Camera's z

World-to-Camera

Indicates that we do the opposite of the stored translation and rotation values

$$p_c = R_c^{-1} T_c^{-1} p_w$$

$$\begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} e_{11} & e_{12} & e_{13} & 0 \\ e_{21} & e_{22} & e_{23} & 0 \\ e_{31} & e_{32} & e_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -c_x \\ 0 & 1 & 0 & -c_y \\ 0 & 0 & 1 & -c_z \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{\text{World-to-Camera Transformation}} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix}$$

Remember to read from right to left

Visualizing in Blender…

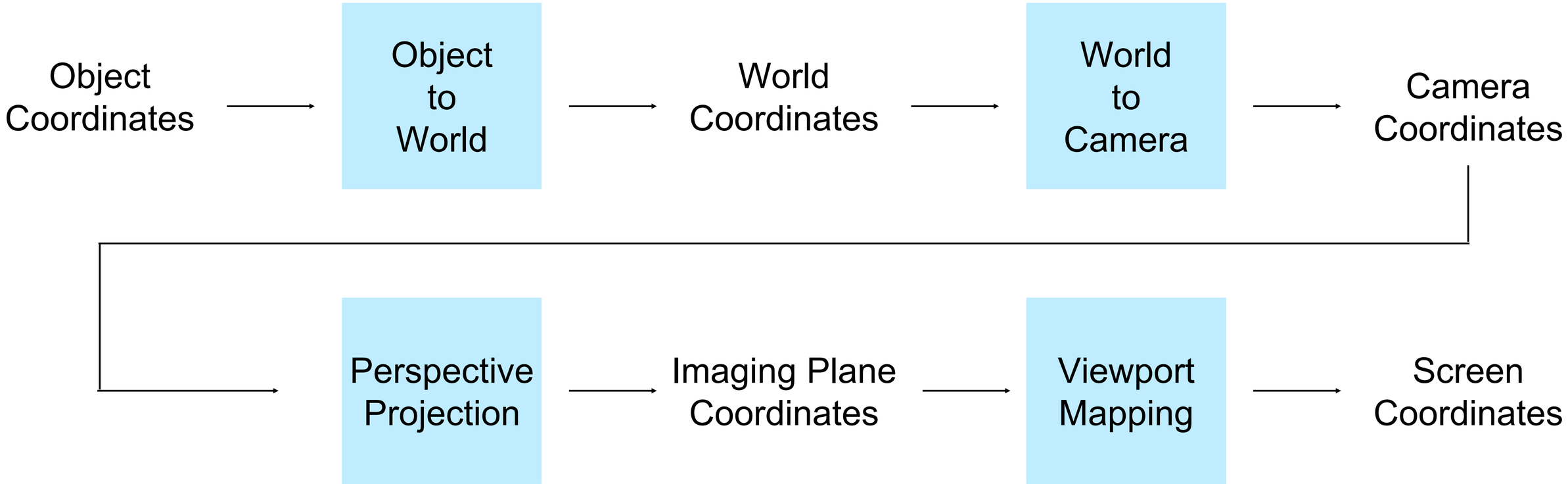
But our points are still 3D, how do we turn them into 2D points on the screen?

We’ll talk about that next time…

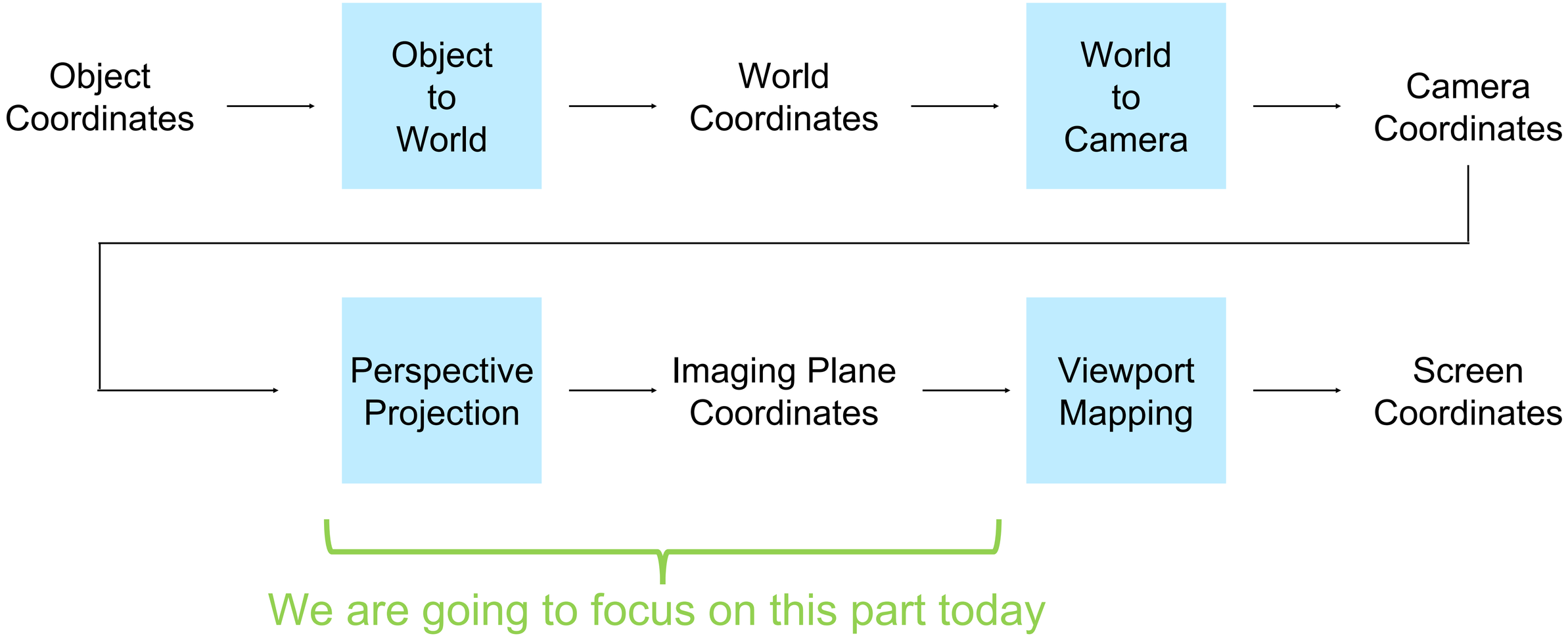
3D Rendering Geometry

CSCI 3800/6050: Computer Graphics

3D Geometry Pipeline



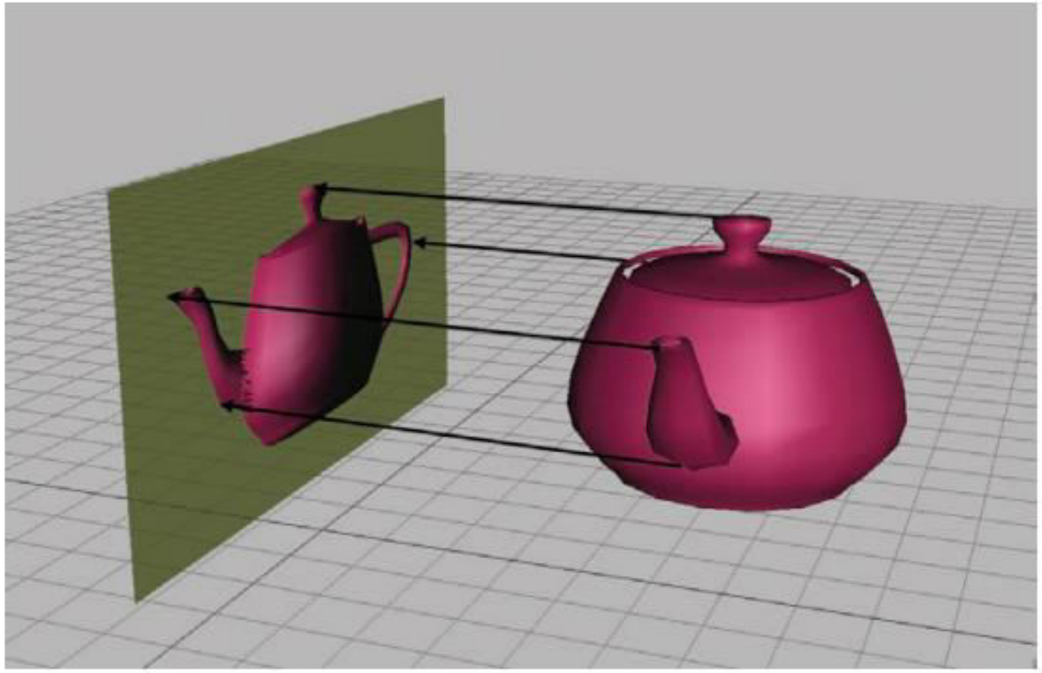
3D Geometry Pipeline



Projection

- To get 2D pictures of a 3D world, you have to use *projection*
  - Orthographic
  - Perspective

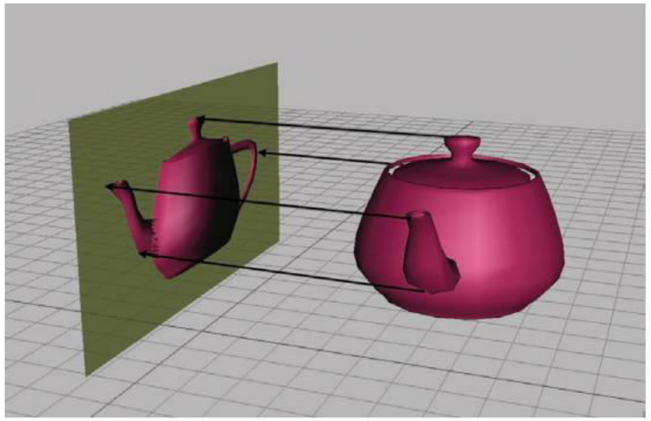
Orthographic Projection



Orthographic projection is just dropping a dimension.

Used in technical drawings, etc.

Orthographic Projection



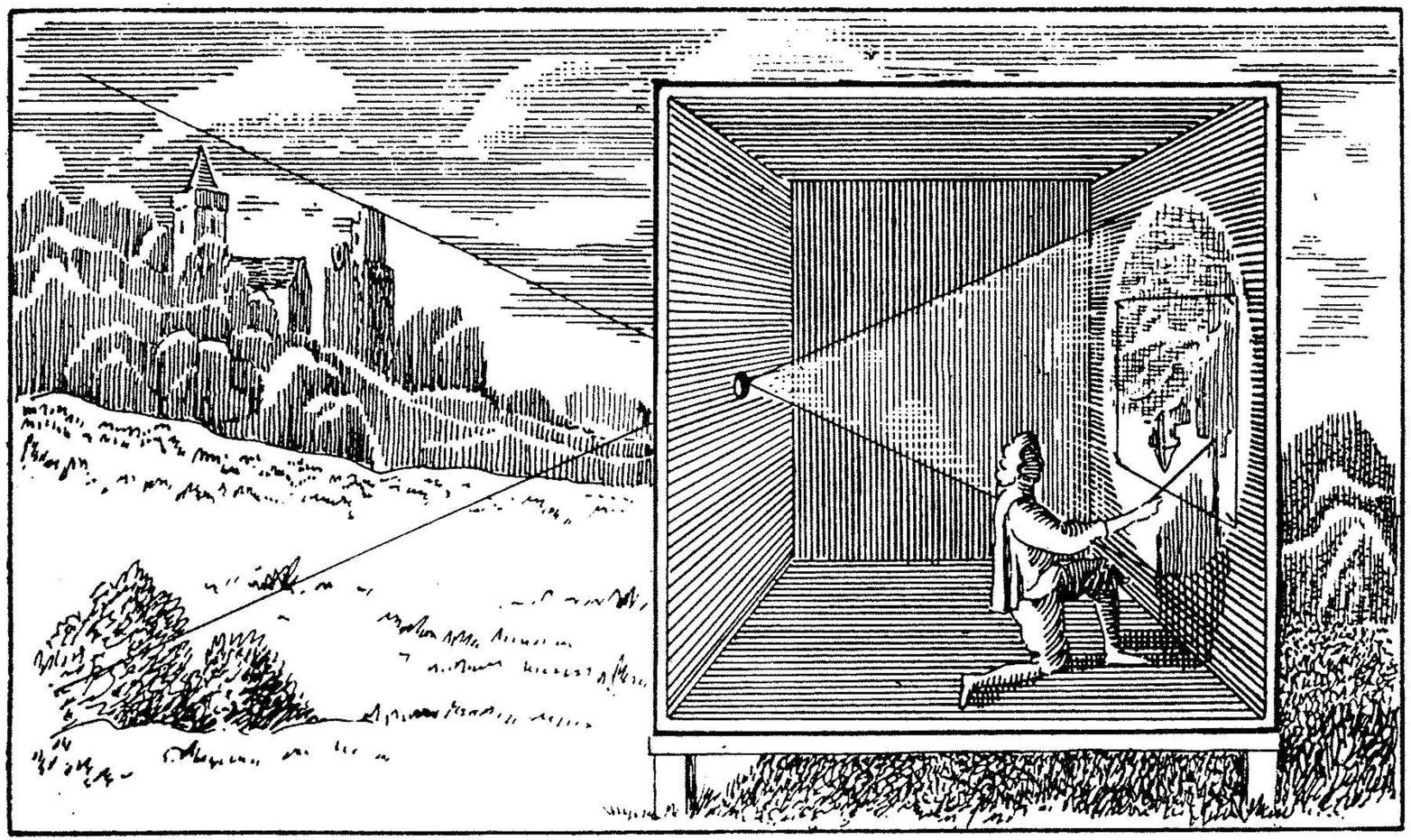
3D point in homogeneous coordinates

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Orthographic projection is just dropping a dimension.

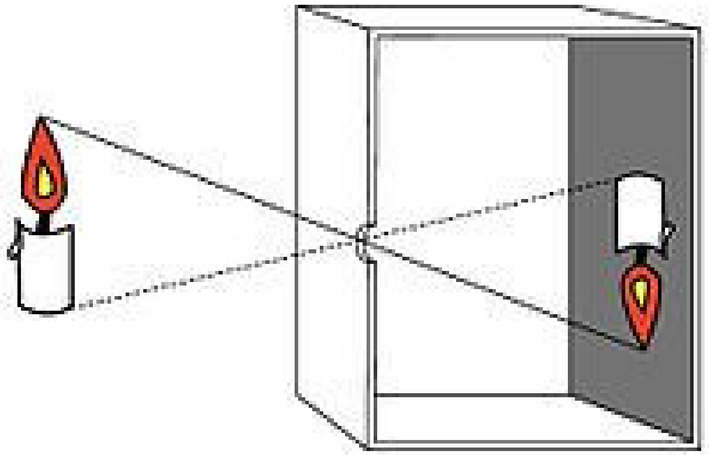
Perspective Projection

Camera Obscura



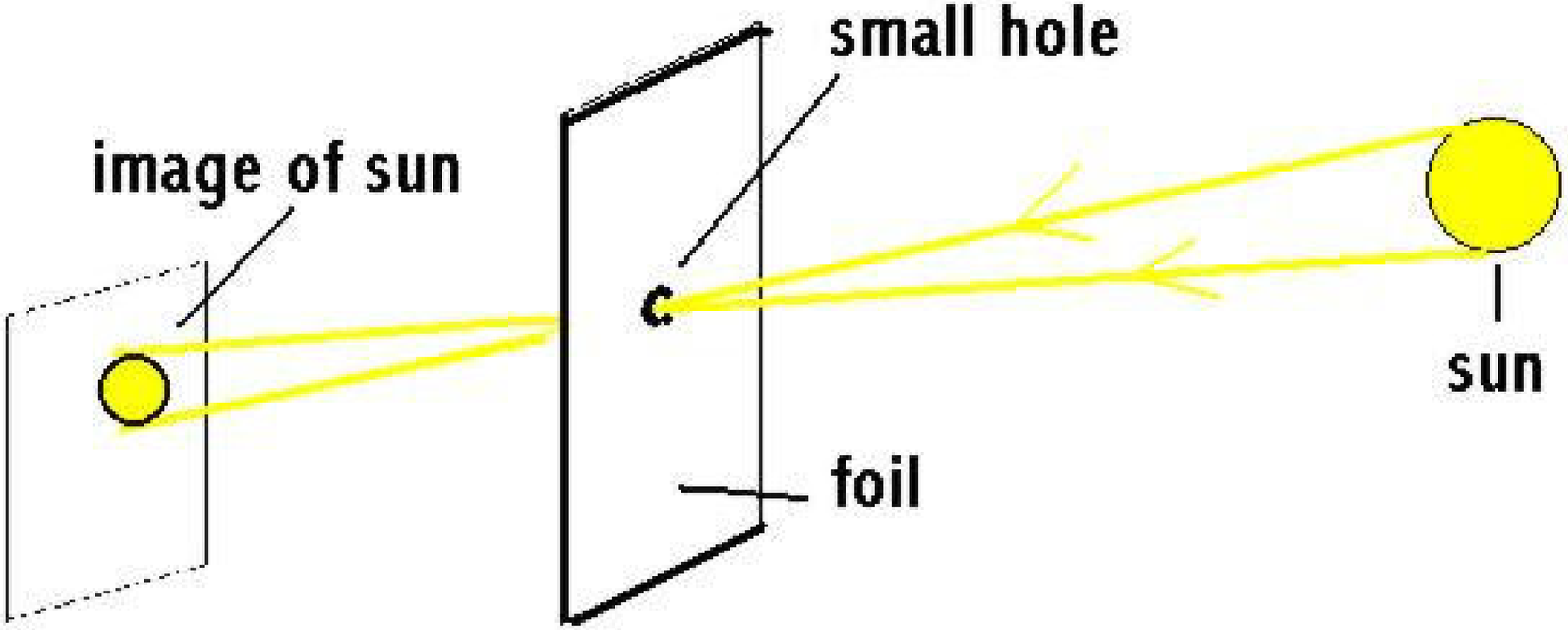
The Pinhole Camera

https://www.bhphotovideo.com/images/images500x500/Santa_Barbara_TEL9RO_4_x_5_Pinhole_151512.jpg



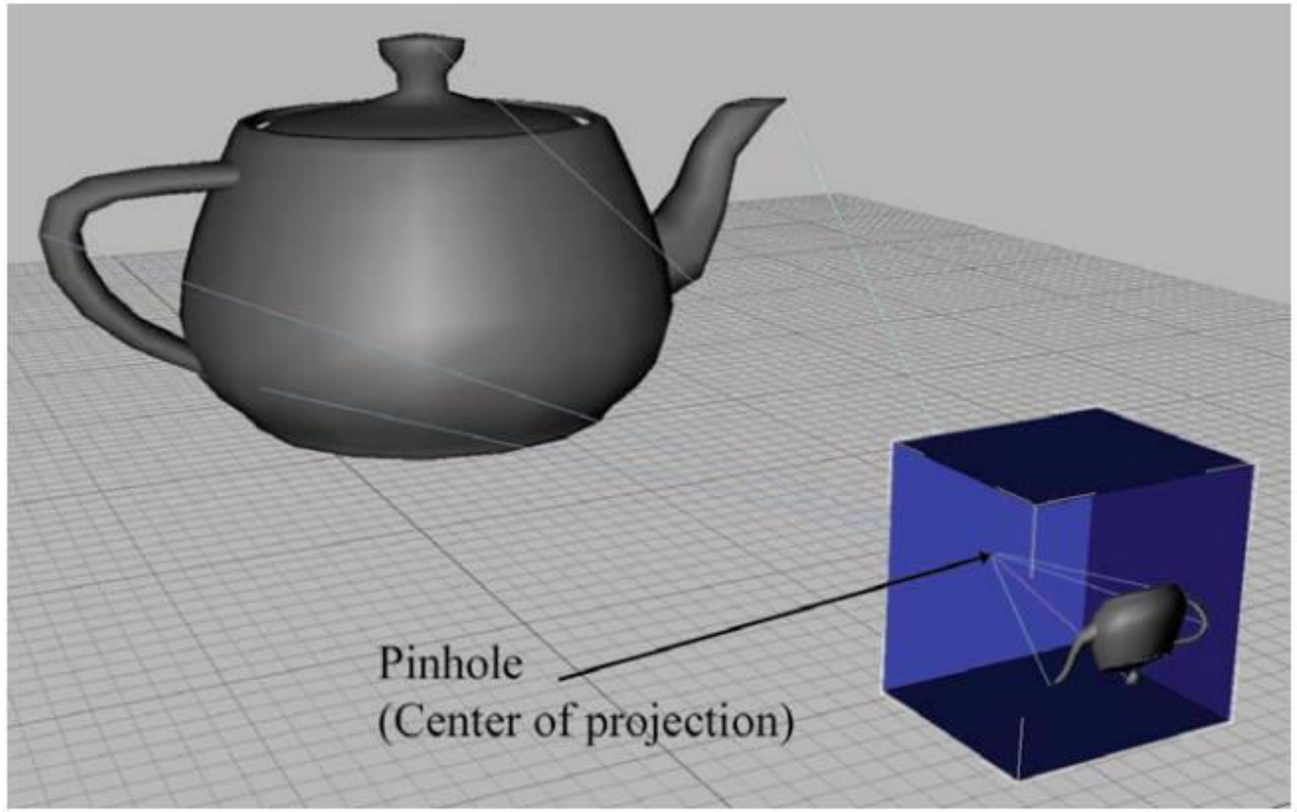
https://upload.wikimedia.org/wikipedia/commons/thumb/7/71/Camera_Oscura.jpg/220px-Camera_Oscura.jpg

Aside: Eclipse Viewing



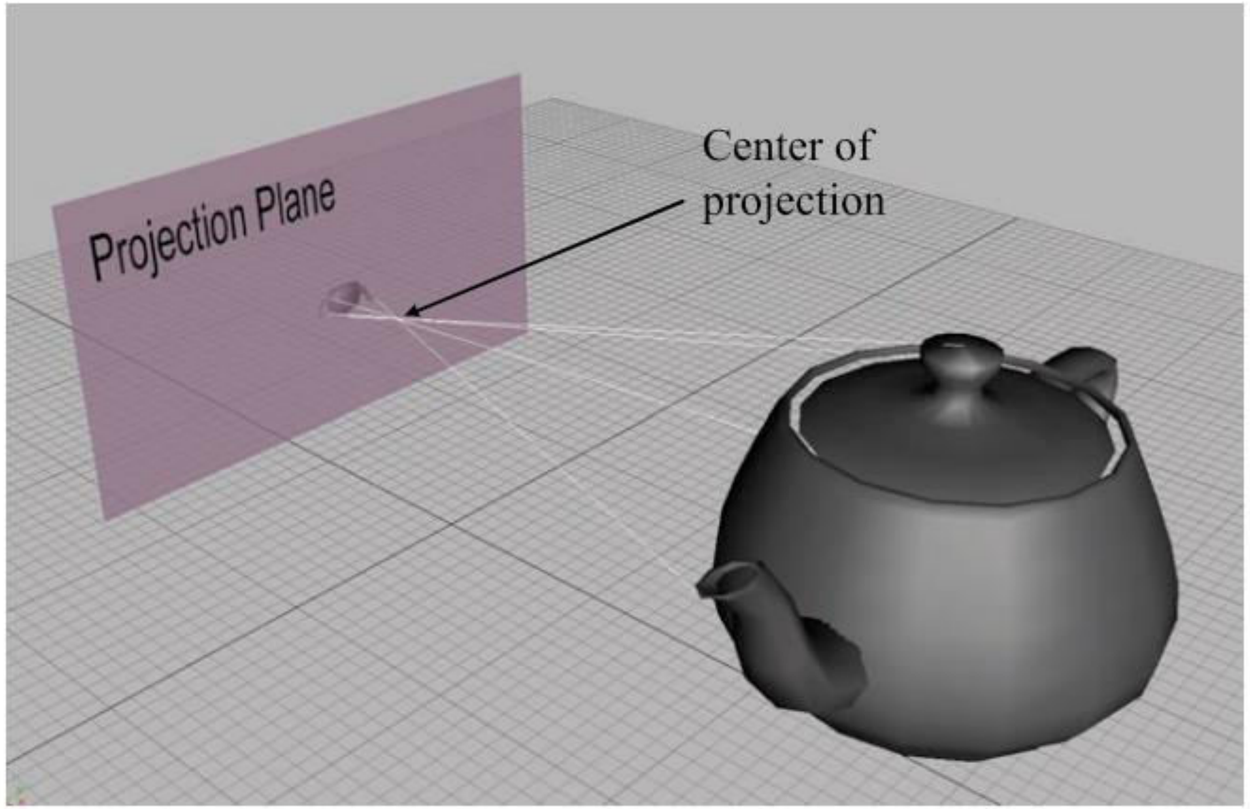
Aside: Eclipse Viewing

Perspective Projection



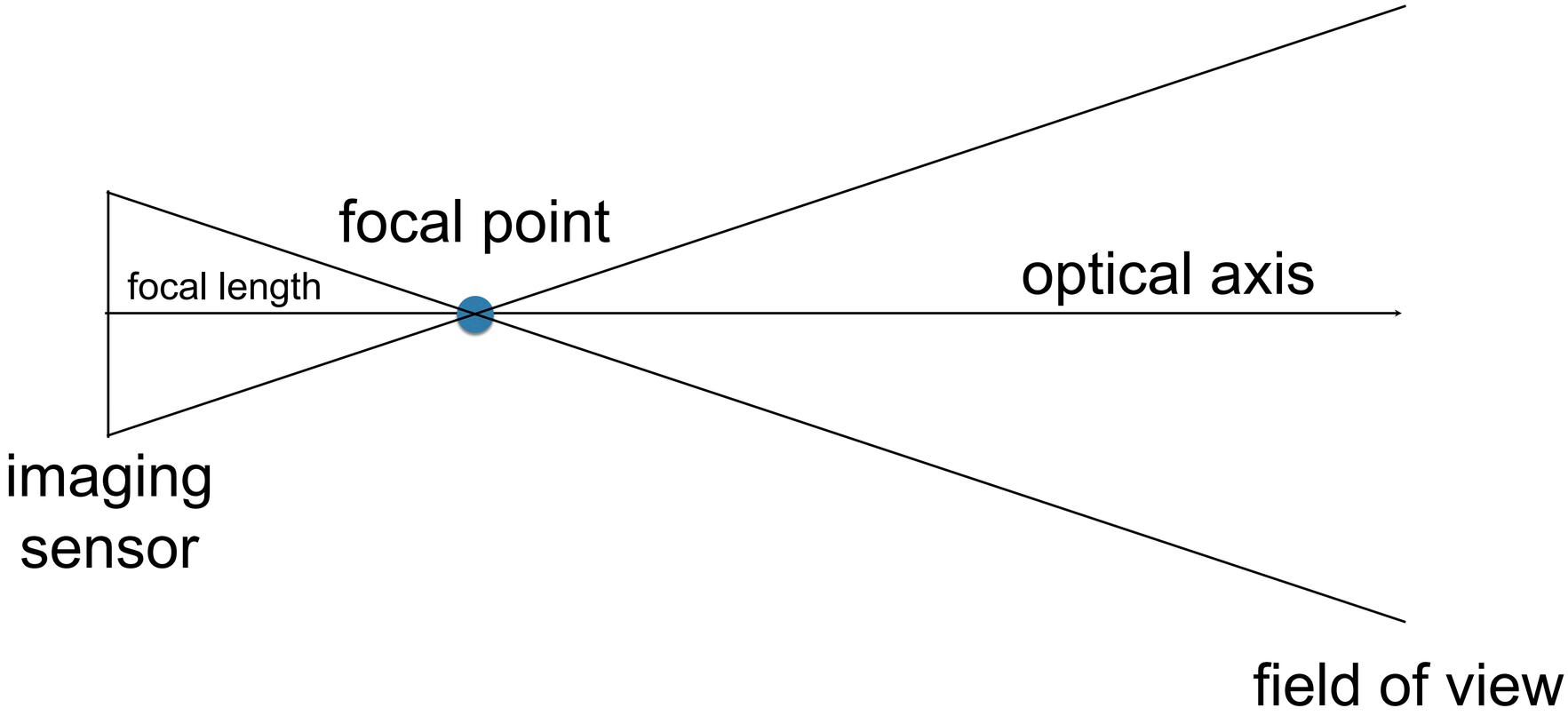
Many graphics systems assume a simple pinhole camera model

Perspective Projection

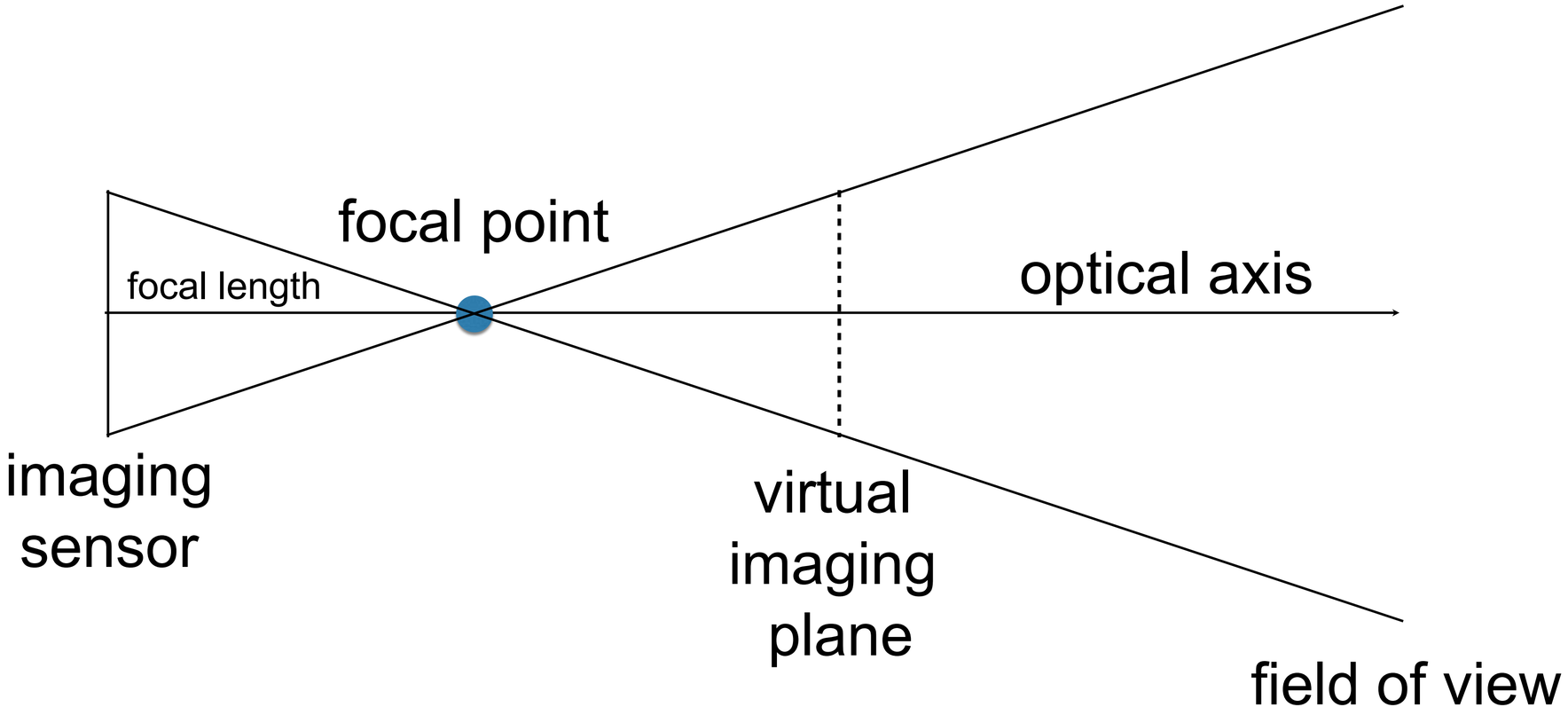


Pinhole camera model

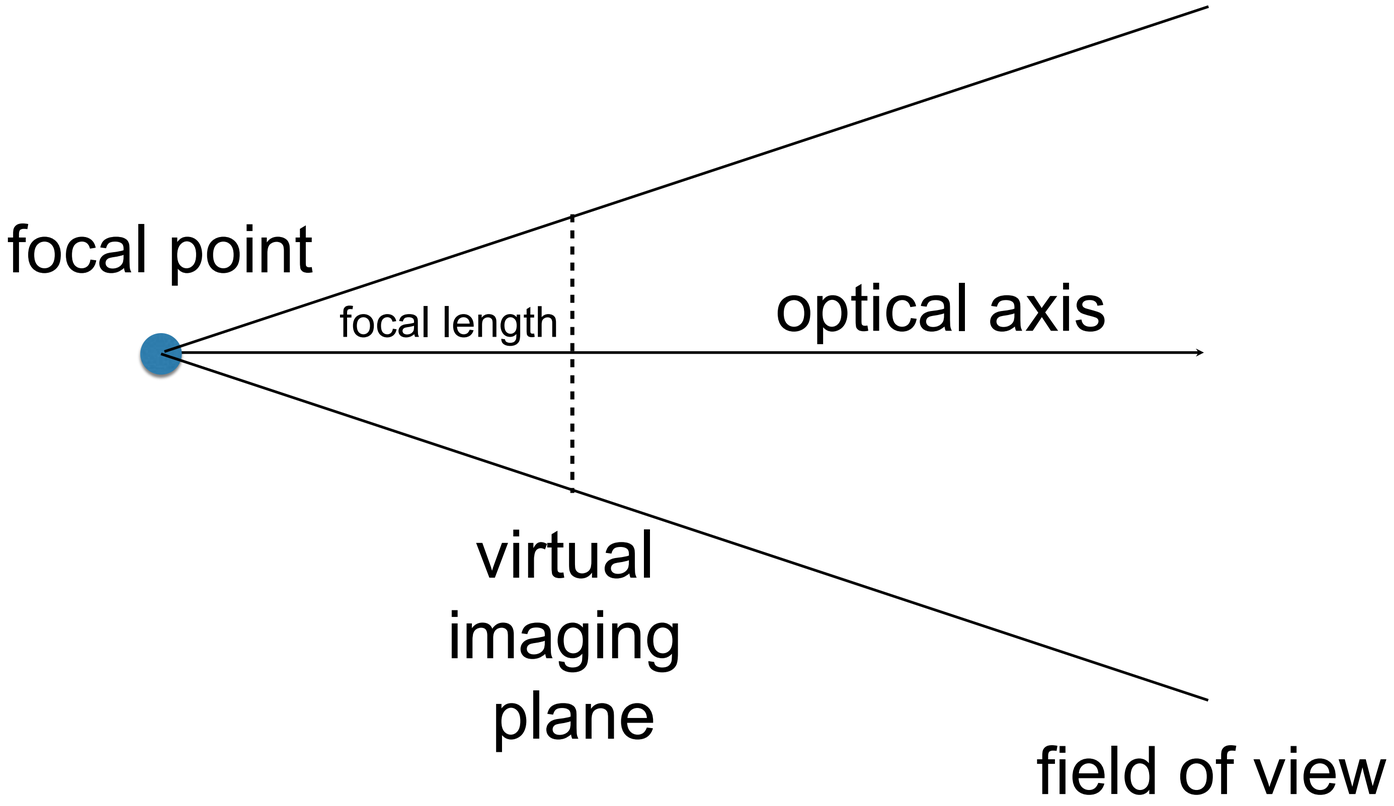
Pinhole Cameras



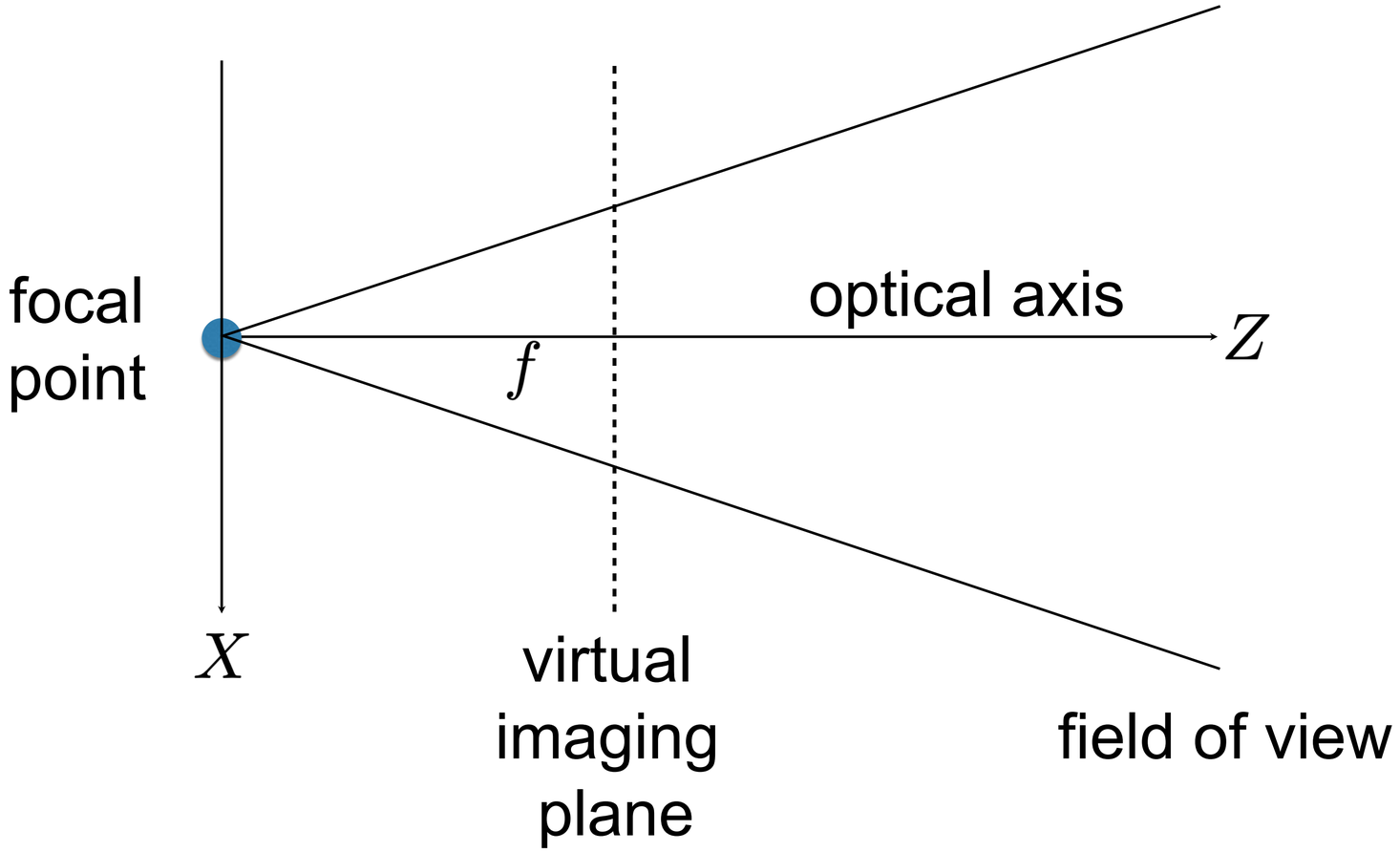
Geometric Model



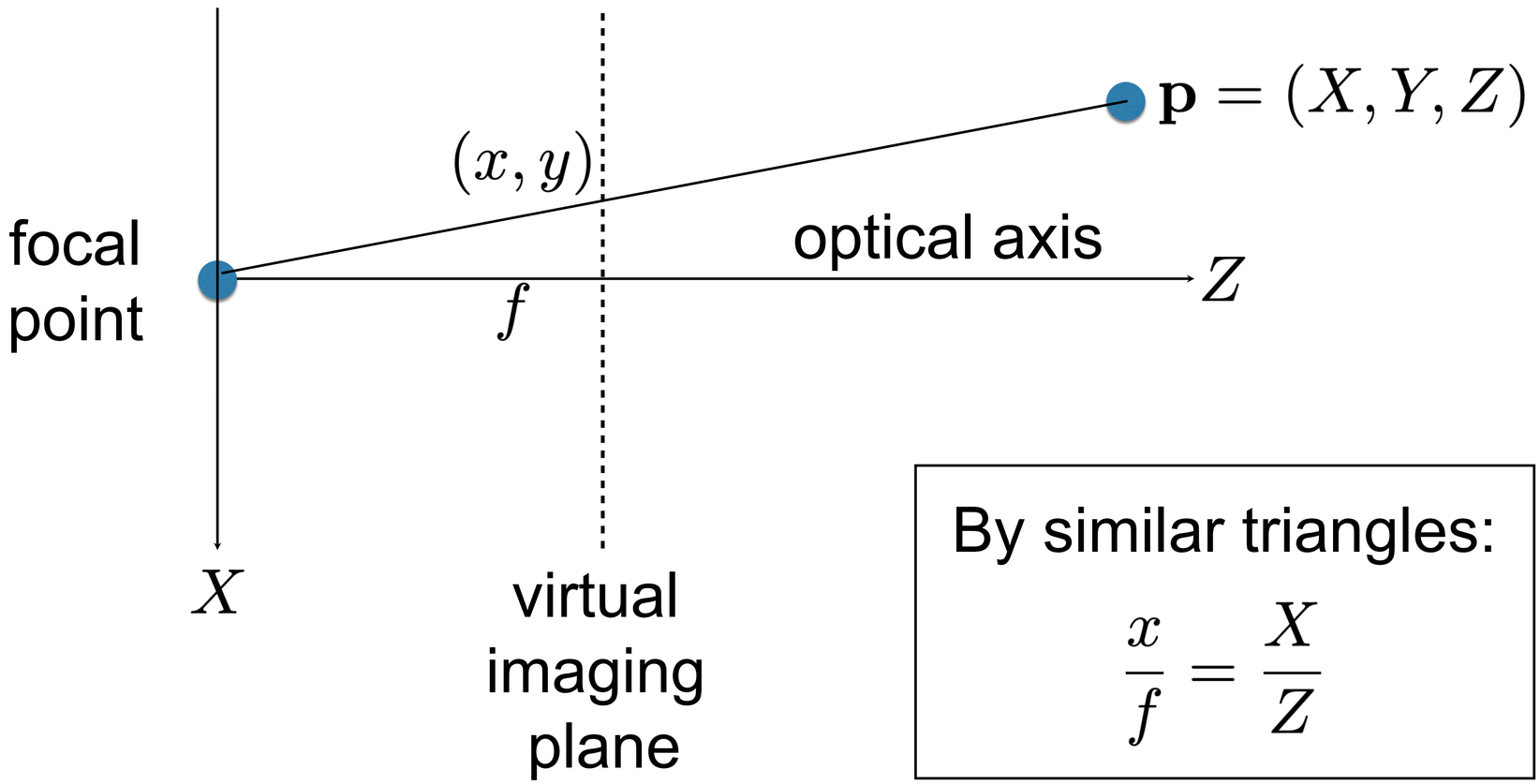
Geometric Model



Camera Coordinates



Projection



By similar triangles:

$$\frac{x}{f} = \frac{X}{Z}$$

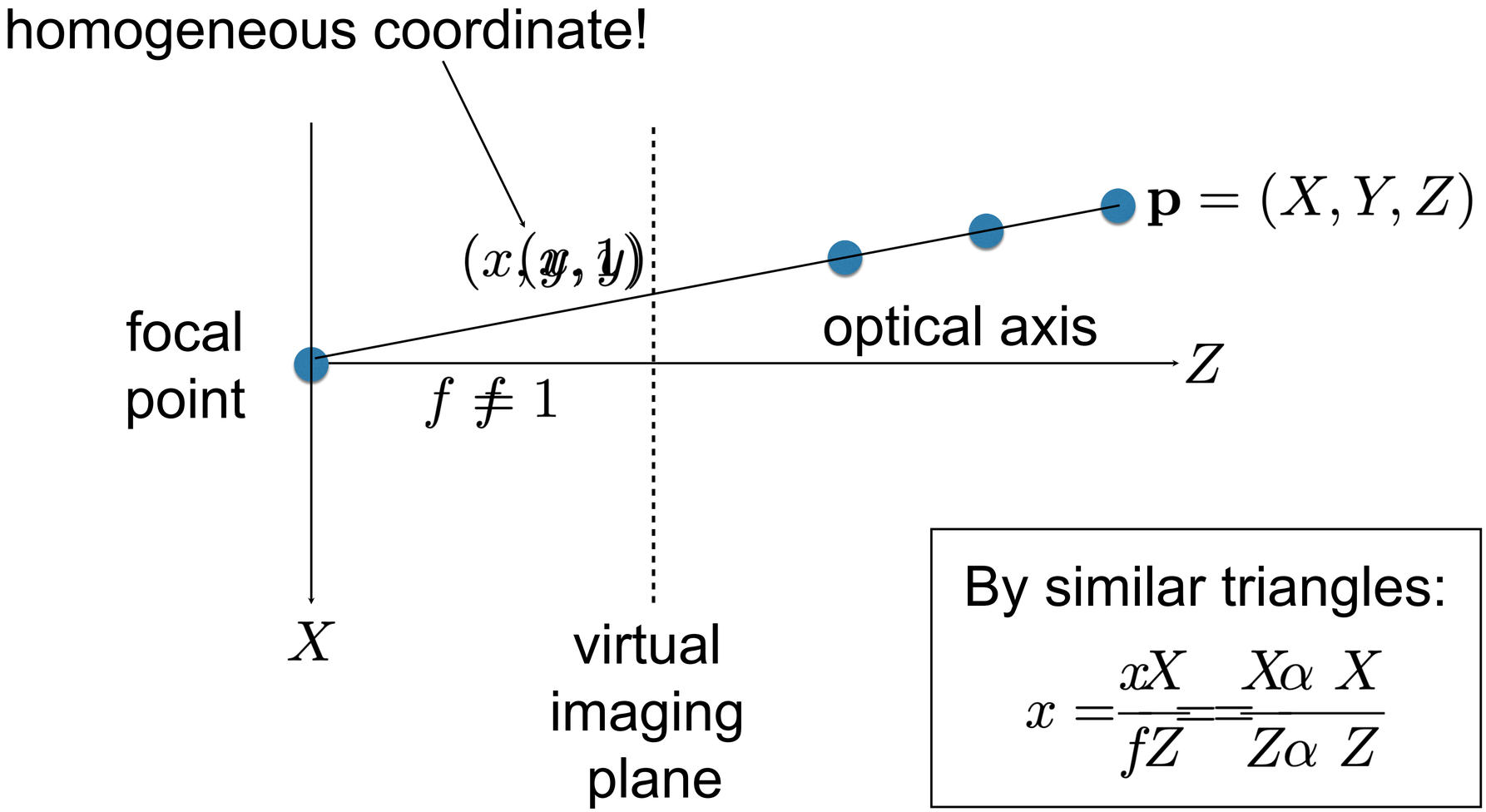
Projection

$$\frac{x}{f} = \frac{X}{Z} \qquad \frac{y}{f} = \frac{Y}{Z}$$

$$(x, y) = \left(\frac{fX}{Z}, \frac{fY}{Z} \right)$$

Note: this is the projected coordinate in real-world units.
To get actual pixel location, scale by pixel density
and apply offset to image origin (more on this later...)

Projection



By similar triangles:

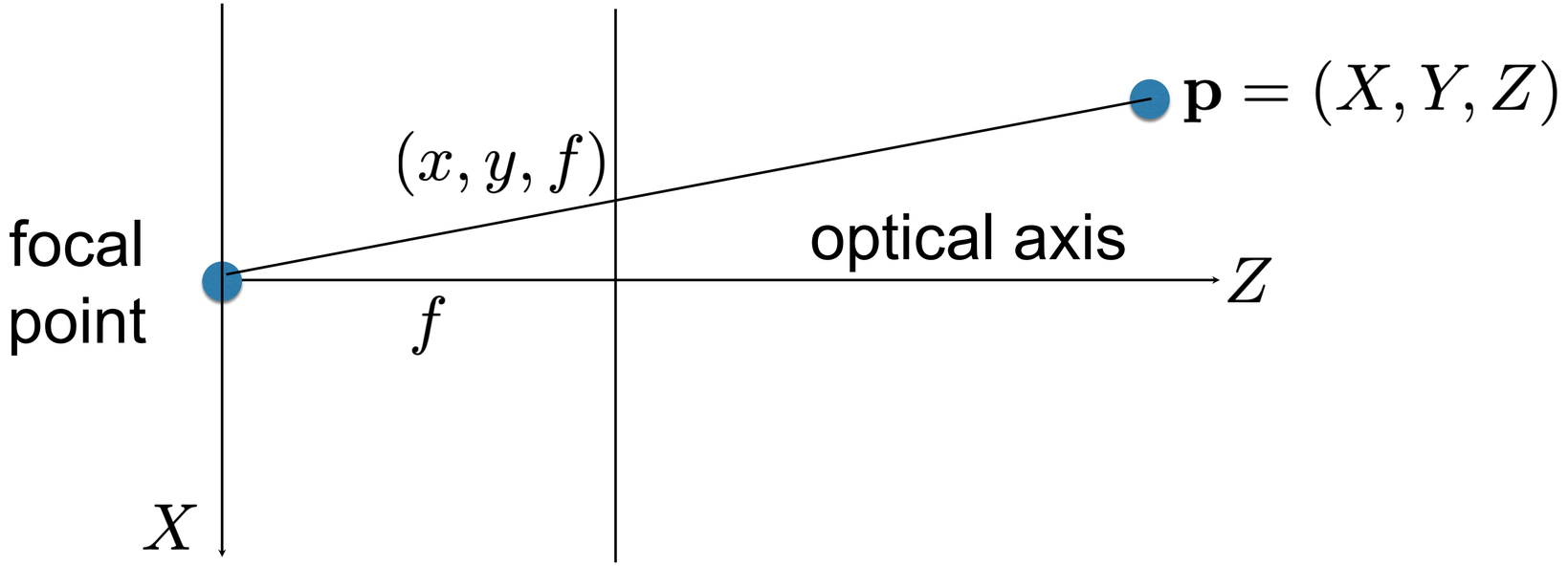
$$x = \frac{X}{Z} = \frac{f}{Z}$$

Key Idea: Homogenous Coordinates

- Homogeneous coordinates are used to represent all 3D points along the ray that falls on the same 2D projection:

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \sim \begin{bmatrix} \alpha\ x \\ \alpha\ y \\ \alpha \end{bmatrix}$$

Perspective Projection



$$\begin{bmatrix} \boxed{\begin{matrix} x \\ y \end{matrix}} \\ f \\ 1 \end{bmatrix} = \begin{bmatrix} fX/Z \\ fY/Z \\ f \\ 1 \end{bmatrix} \sim \begin{bmatrix} X \\ Y \\ Z \\ Z/f \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \boxed{1/f} & 0 \end{bmatrix} \begin{bmatrix} \boxed{\begin{matrix} X \\ Y \\ Z \\ 1 \end{matrix}} \end{bmatrix}$$

The Homogenous Divide

$$\begin{bmatrix} x \\ y \\ f \\ 1 \end{bmatrix} = \begin{bmatrix} fX/Z \\ fY/Z \\ f \\ 1 \end{bmatrix} \sim \begin{bmatrix} X \\ Y \\ Z \\ Z/f \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/f & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

What does this symbol mean?

The Homogenous Divide

- Multiplying by a 4 x 4 projection matrix no longer guarantees that a 1 will remain in the homogenous coordinate.
- In other words, a projection transformation will give us a 3D point that is usually not on the image plane.
- To solve this, reproject the point back to the image plane by dividing by the homogenous value.

Homogenous Divide (conceptually)

$$\begin{bmatrix} x/w \\ y/w \\ z/w \\ 1 \end{bmatrix} \sim \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

Alternative Form

One way (some implementation advantages):

$$\begin{bmatrix} x \\ y \\ f \\ 1 \end{bmatrix} = \begin{bmatrix} fX/Z \\ fY/Z \\ f \\ 1 \end{bmatrix} \sim \begin{bmatrix} X \\ Y \\ Z \\ Z/f \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/f & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Another way (some conceptual advantages):

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} fX/Z \\ fY/Z \\ 1 \end{bmatrix} \sim \begin{bmatrix} X \\ Y \\ Z/f \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1/f & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Putting It Together With Projection

$$\begin{bmatrix} \boxed{\begin{matrix} x \\ y \end{matrix}} \\ f \\ 1 \end{bmatrix} \sim \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ Z_c/f \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/f & 0 \end{bmatrix} \underbrace{\begin{bmatrix} e_{11} & e_{12} & e_{13} & 0 \\ e_{21} & e_{22} & e_{23} & 0 \\ e_{31} & e_{32} & e_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -c_x \\ 0 & 1 & 0 & -c_y \\ 0 & 0 & 1 & -c_z \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{\text{World-to-camera transformation}} \begin{bmatrix} \boxed{\begin{matrix} X_w \\ Y_w \\ Z_w \end{matrix}} \\ 1 \end{bmatrix}$$

Normalize Project Rotate Translate

A Better Projection Matrix

- If your camera setup uses field-of-view rather than focal length, then you can use this projection matrix.
- This is the projection matrix you will use for lab 3.
- It has other benefits that we will talk about next time.

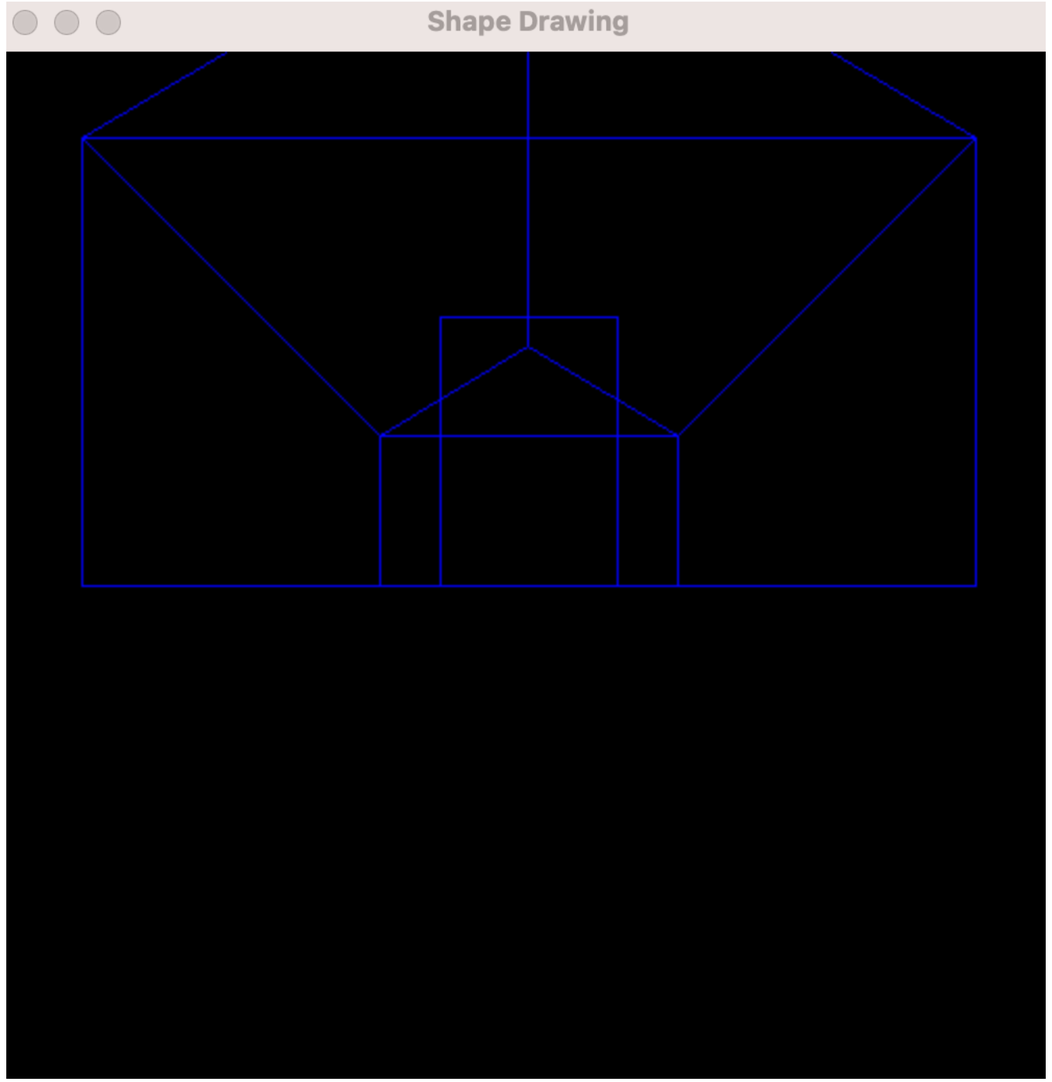
$$\begin{bmatrix} \text{zoom}_x & 0 & 0 & 0 \\ 0 & \text{zoom}_y & 0 & 0 \\ 0 & 0 & \frac{f+n}{f-n} & \frac{-2nf}{f-n} \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

$$\text{zoom} = \frac{1}{\tan(\text{fov}/2)}$$

n = near plane distance
f = far plane distance

Let’s try it in NumPy…

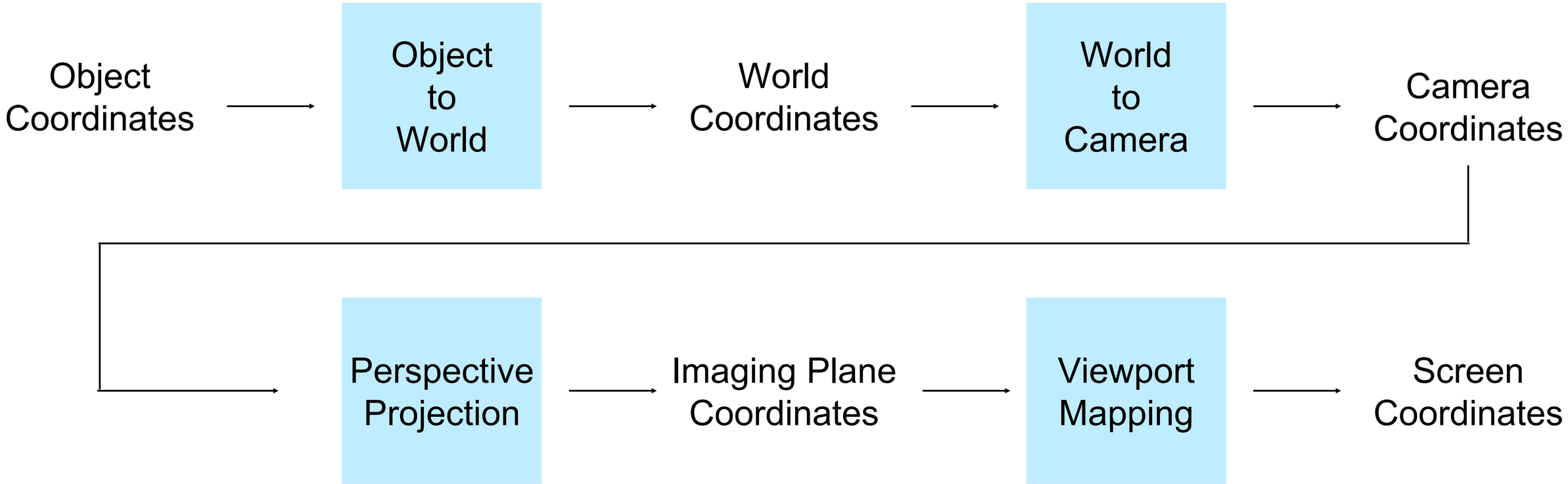
Lab 3 Intro



Clipping Tests and Viewing

CSCI 3800/6820: Computer Graphics

3D Geometry Pipeline



Let’s revisit the pipeline...

Pipeline So Far

Idea: let's cull as much as we can before dividing

World-to-camera transformation

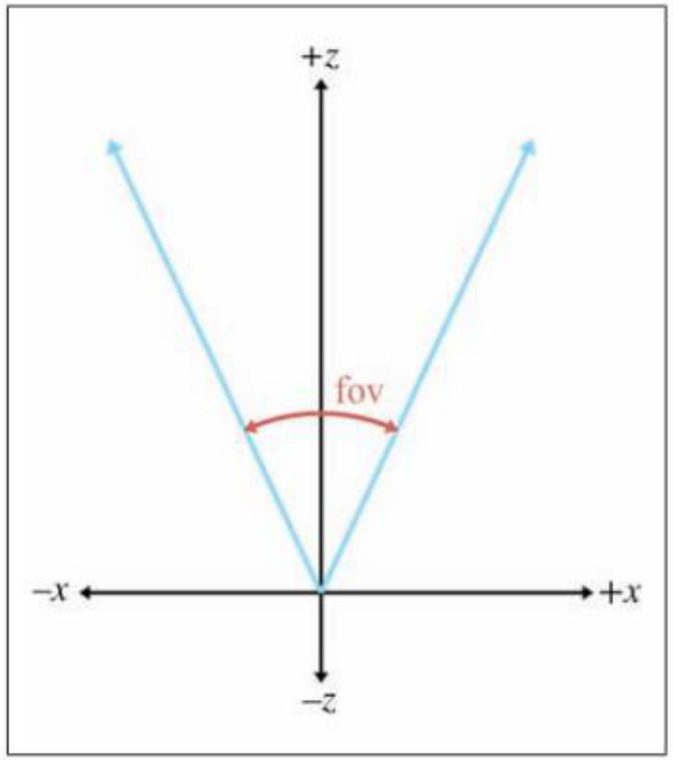
$$\begin{bmatrix} \boxed{\begin{matrix} x \\ y \end{matrix}} \\ f \\ 1 \end{bmatrix} \sim \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ Z_c/f \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/f & 0 \end{bmatrix} \begin{bmatrix} e_{11} & e_{12} & e_{13} & 0 \\ e_{21} & e_{22} & e_{23} & 0 \\ e_{31} & e_{32} & e_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -c_x \\ 0 & 1 & 0 & -c_y \\ 0 & 0 & 1 & -c_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \boxed{\begin{matrix} X_w \\ Y_w \\ Z_w \end{matrix}} \\ 1 \end{bmatrix}$$

Normalize Project Rotate Translate

Big problem: lots of time spent on stuff you can't see!

Field of View

- All cameras have a limited field of view
- Field of view depends on the focal length
  - Zoomed in - smaller
  - Zoomed out - larger



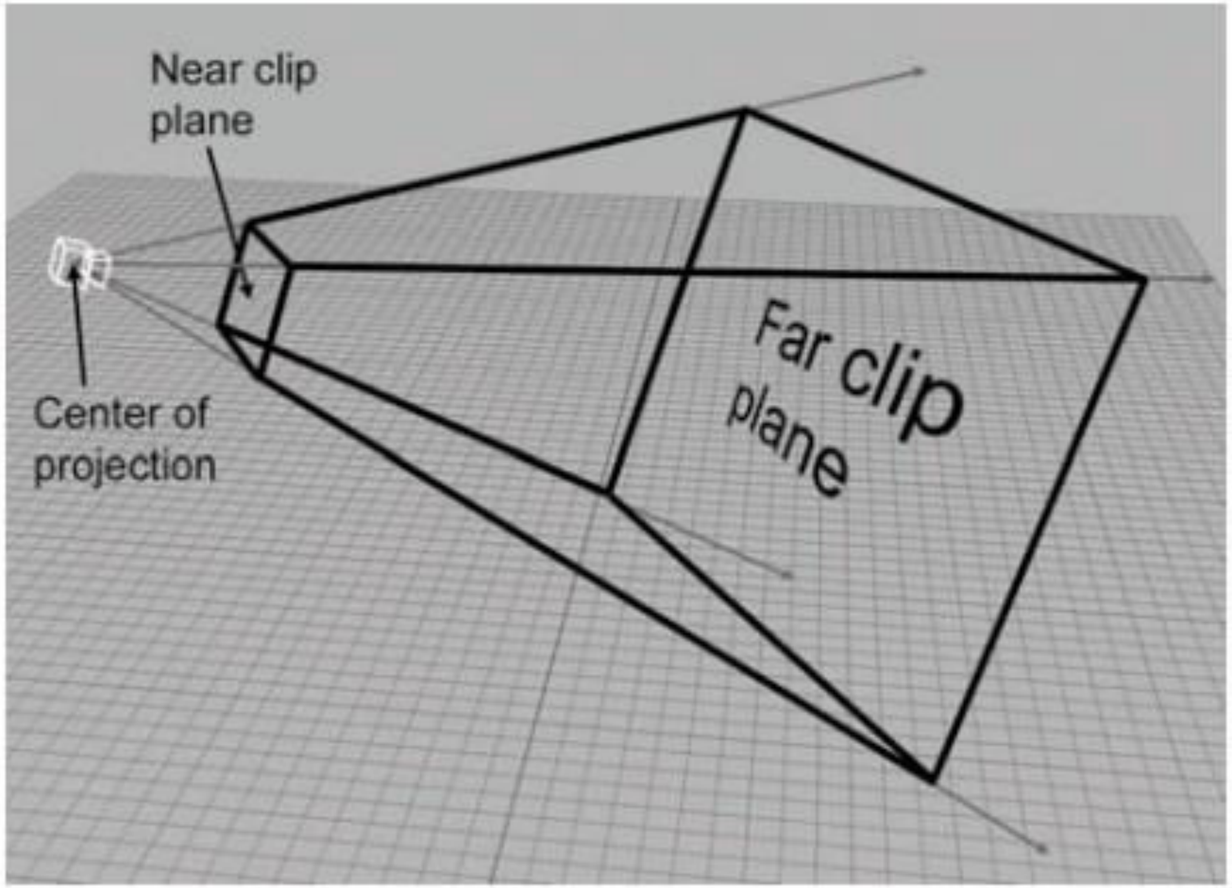
Near and Far Planes

- We don’t want to render things behind us, or perhaps even just barely in front of us
- We don’t care about things too far away to see well

“Near plane”

“Far plane”

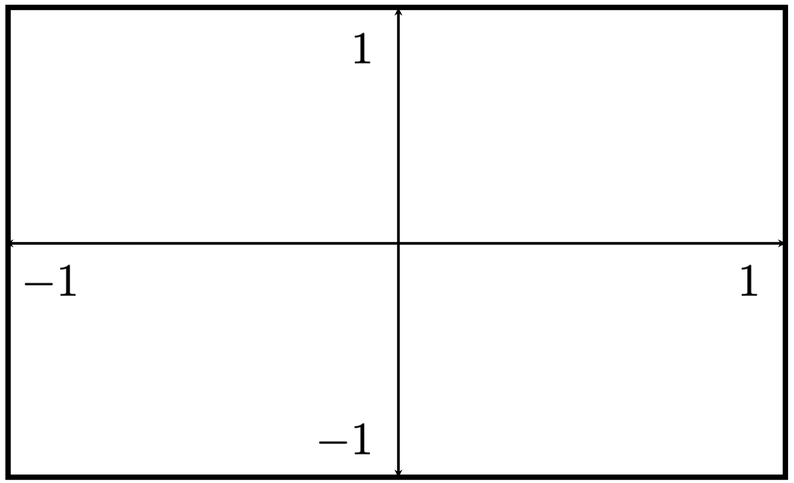
View Frustum



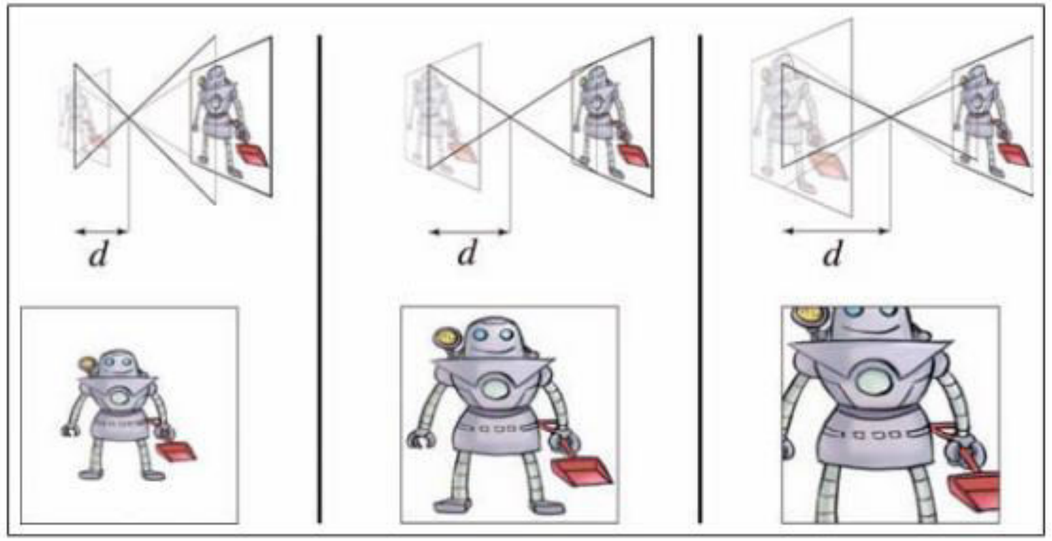
Can we clip things outside the view frustum without doing a divide?

Canonical View

- To simplify, let's assume we map to [-1,1] in both *x* and *y* directions
- Also map [near, far] depth range to [-1,1]
- Maps frustum to $[-1,1]^3$ cube



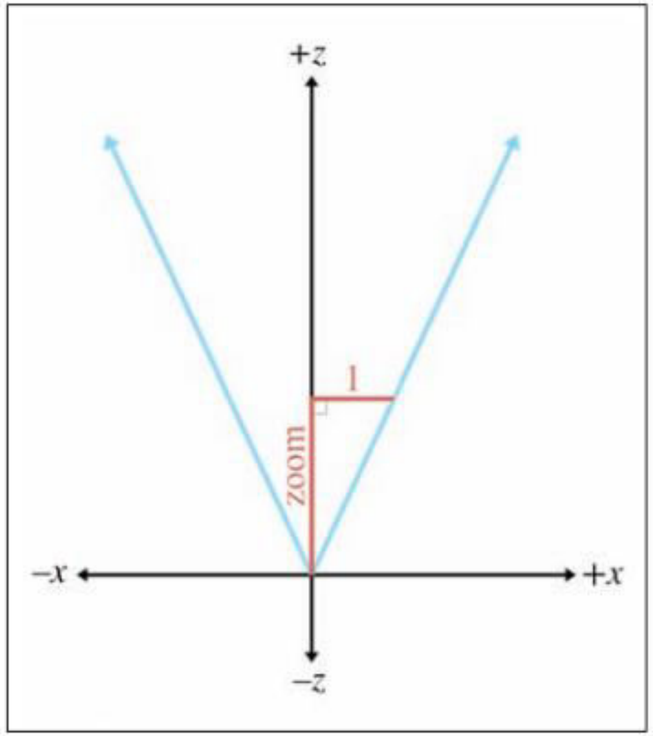
Changing Focal Length



Changing focal length changes overall zoom,
but also affects the shape of the view frustum

Zoom

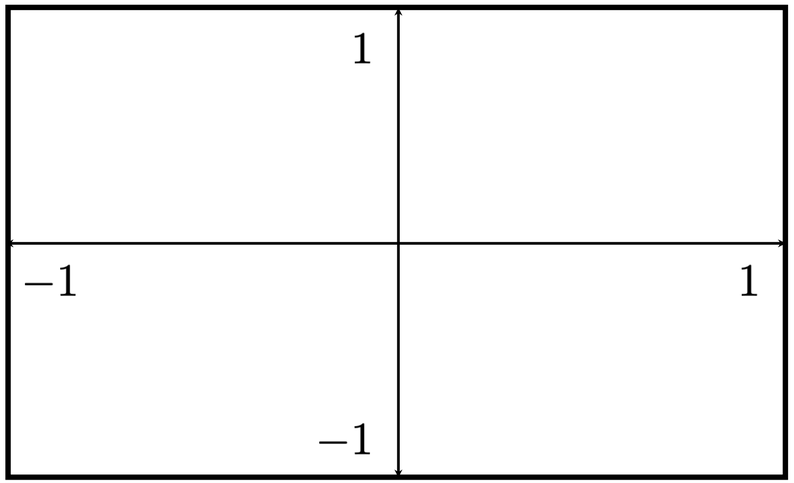
- Mapping to a canonical window loses
  - Real horizontal width
  - Real vertical width
- We'll need to fold this into our projection matrix
- Think in terms of different "zoom" levels for x and y



$$\text{zoom} = \frac{1}{\tan(\text{fov}/2)}$$

The Clip Matrix

- Let’s build a new projection matrix that
  - Scales visible x to [-1,1]
  - Scales visible y to [-1,1]
  - Scales near to far z to [-1,1]



The Clip Matrix

- Let's build a new projection matrix that
  - Scales visible x to [-1,1]
  - Scales visible y to [-1,1]
  - Scales near to far z to [-1,1]

$$\begin{bmatrix} \text{zoom}_x & 0 & 0 & 0 \\ 0 & \text{zoom}_y & 0 & 0 \\ 0 & 0 & \frac{f+n}{f-n} & \frac{-2nf}{f-n} \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

$$\text{zoom} = \frac{1}{\tan(\text{fov}/2)}$$

n = near plane distance
f = far plane distance

The Clip Matrix

$$\begin{bmatrix} \boxed{\begin{matrix} x/w \\ y/w \\ z/w \end{matrix}} \\ 1 \end{bmatrix} \sim \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} \text{zoom}_x & 0 & 0 & 0 \\ 0 & \text{zoom}_y & 0 & 0 \\ 0 & 0 & \frac{f+n}{f-n} & \frac{-2nf}{f-n} \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix}$$

All of these are in the range [-1,1] for things in view

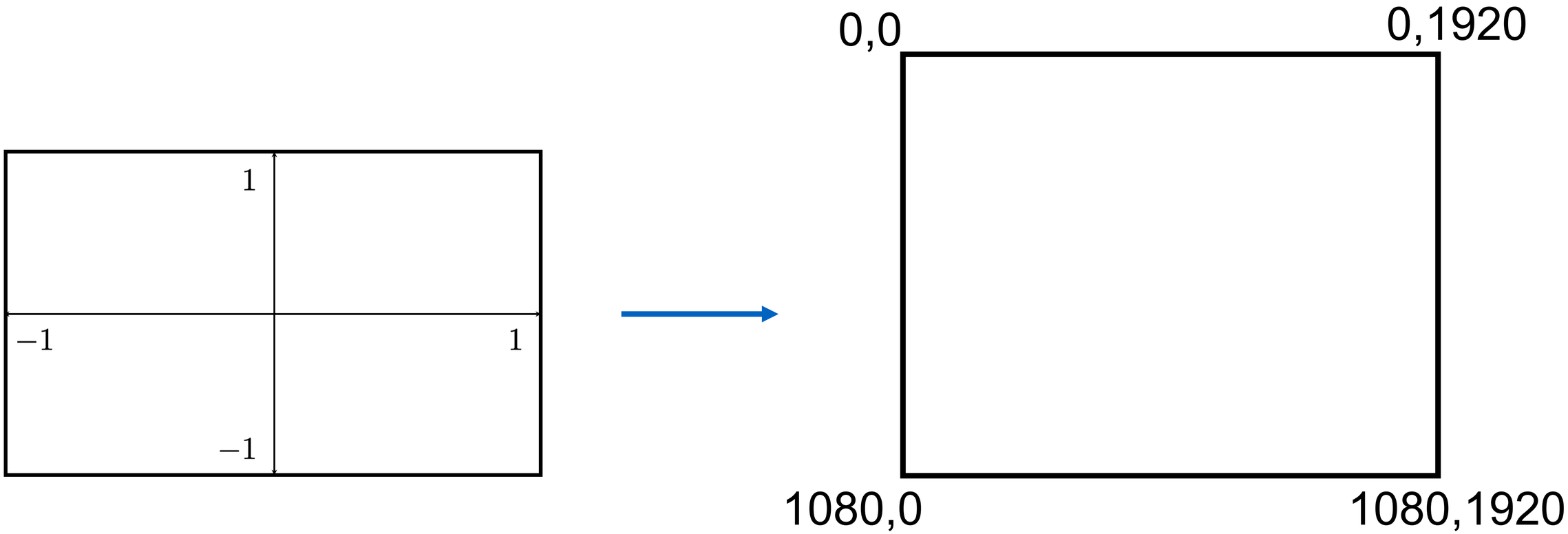
Clipping Tests

Left	$x < -w$
Right	$x > w$
Bottom	$y < -w$
Top	$y > w$
Near	$z < -w$
Far	$z > w$

Culling / Clipping

- If an entire primitive (line, polygon) fails the same clipping test, it is outside the field of view—if so, throw out
- If part fails and part passes, clip to the portion in view (create partial primitive) and process from there
- Clipping against multiple planes may not leave anything left — if so, throw out

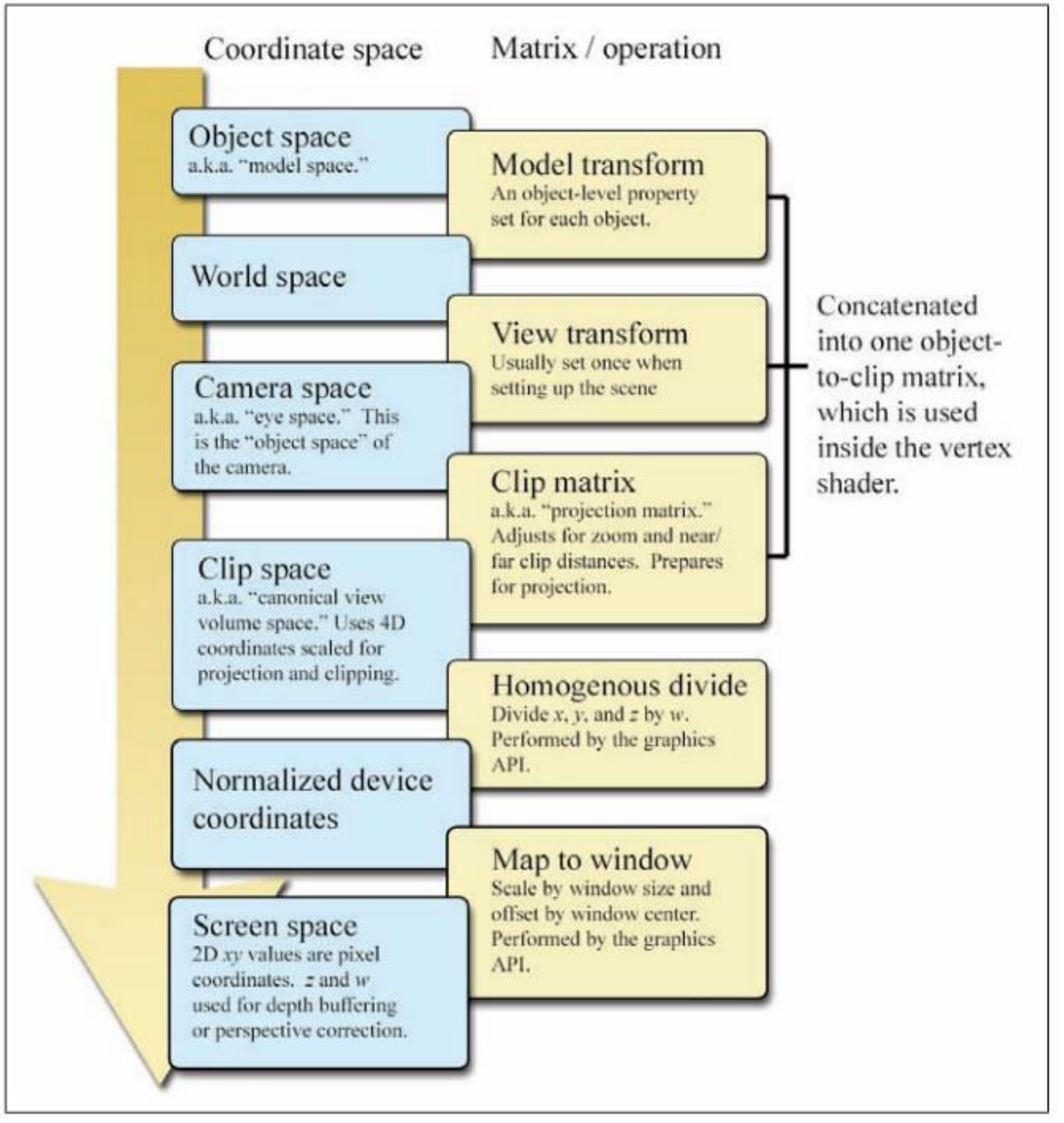
To Screen Space



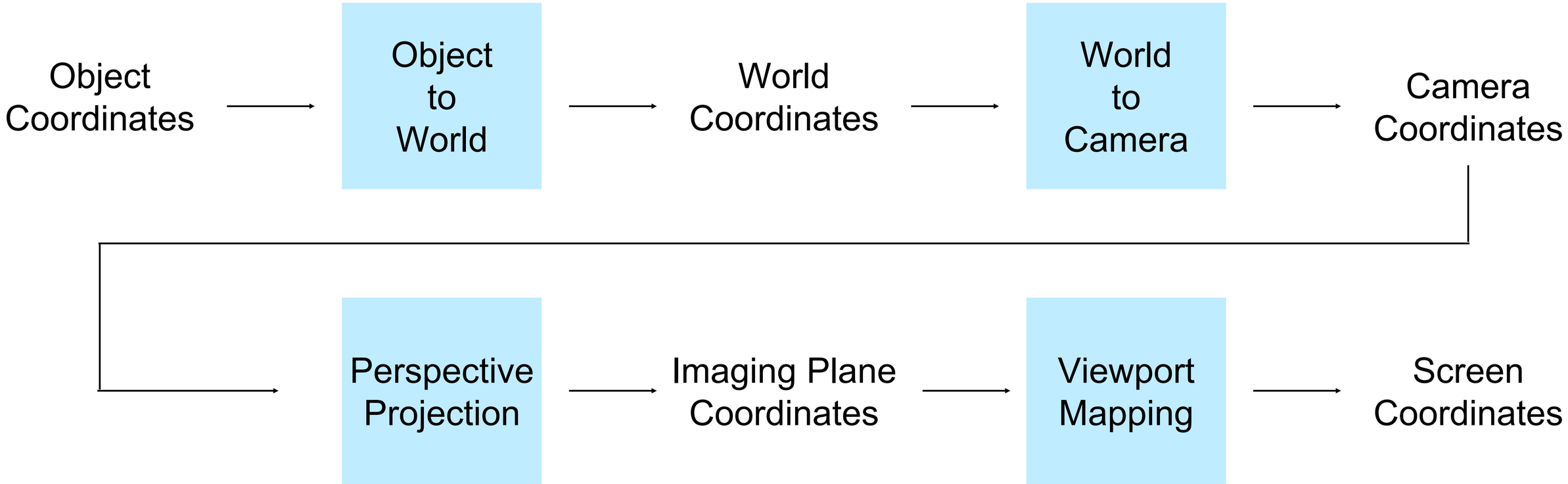
To Screen Space

- Map [-1,1] x [-1,1] to screen
  - Scale x by half the width
  - Invert y and scale by half the height
  - Translate origin from center to upper left corner
- Sometimes called the “Viewport Transformation”

$$\begin{bmatrix} x_{\text{screen}} \\ y_{\text{screen}} \\ 1 \end{bmatrix} = \begin{bmatrix} \text{width}/2 & 0 & \text{width}/2 \\ 0 & -\text{height}/2 & \text{height}/2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x/w \\ y/w \\ 1 \end{bmatrix}$$


Coordinate space
Matrix / operation
Object space
a.k.a. "model space."
Model transform
An object-level property set for each object.
World space
View transform
Usually set once when setting up the scene
Camera space
a.k.a. "eye space." This is the "object space" of the camera.
Clip matrix
a.k.a. "projection matrix." Adjusts for zoom and near/far clip distances. Prepares for projection.
Concatenated into one object-to-clip matrix, which is used inside the vertex shader.
Clip space
a.k.a. "canonical view volume space." Uses 4D coordinates scaled for projection and clipping.
Homogenous divide
Divide x, y, and z by w. Performed by the graphics API.
Normalized device coordinates
Map to window
Scale by window size and offset by window center. Performed by the graphics API.
Screen space
2D xy values are pixel coordinates. z and w used for depth buffering or perspective correction.

3D Geometry Pipeline



Rendering Geometry

- ✓ Transform from object to world coordinates
- ✓ Transform from world to camera coordinates
- ✓ Project to clip space
- ✓ Perform clipping: near plane, far plane, field of view
- ✓ Perform homogenous divide (map to Canonical Space)
- ✓ Transform from canonical to screen coordinates

Let’s go through a full example…