Course Summary

EDAF80
Michael Doggett

Slides by Jacob Munkberg 2012-13

2022

Today

- What you need to know for the exam
- Questions

Extra labs next week

- Friday October 21
  - 10-12 and 13-15

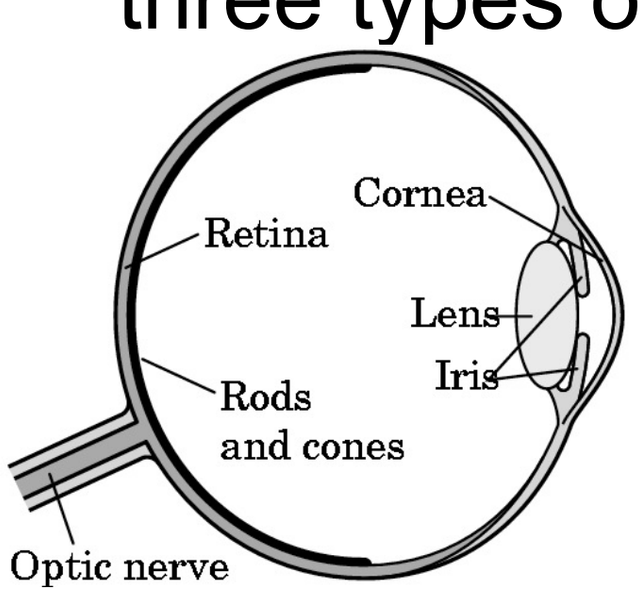
Exam

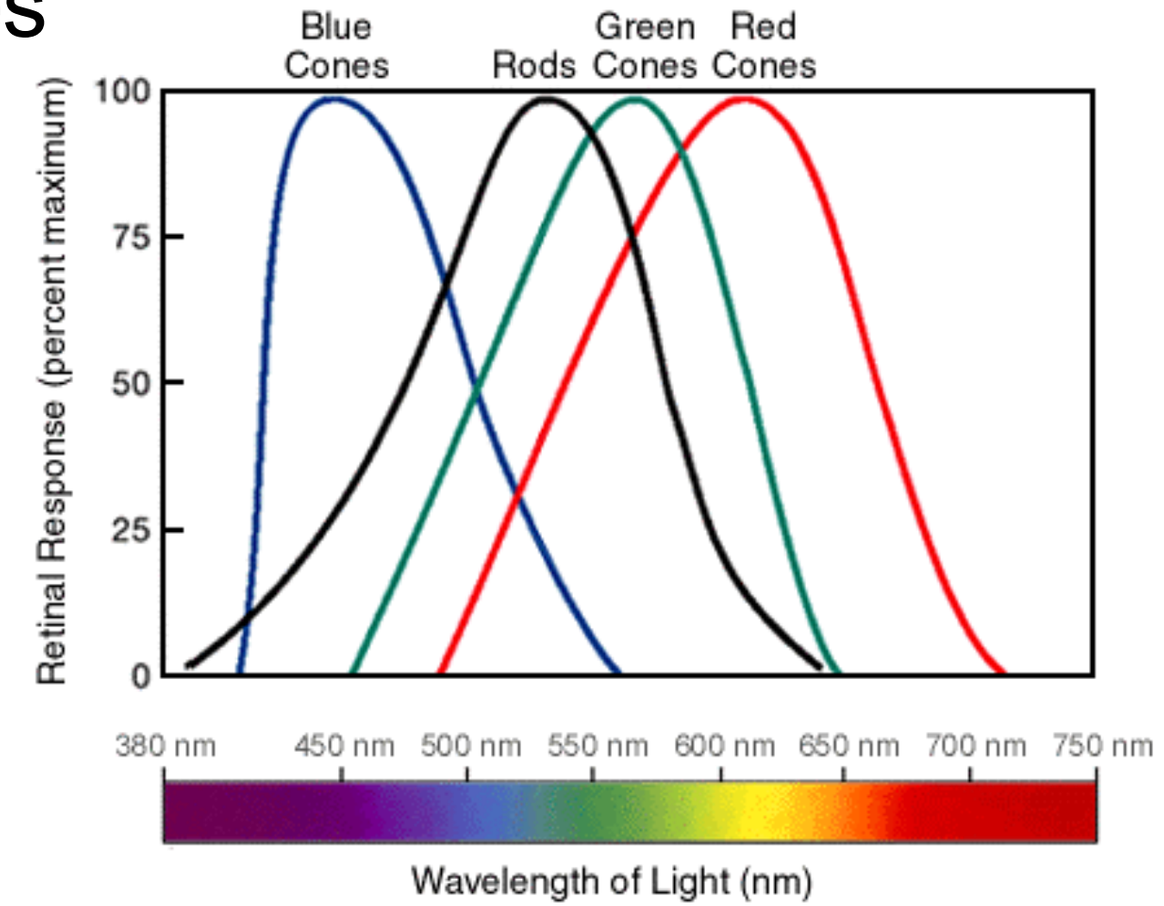
- Thursday October 27: 08-13
- Location: **EDEN022, EDEN025**
  - Allhelgona kyrkogata 14
- No electronic calculator, books or notes
- Example old exams (with solutions)
  - http://cs.lth.se/edaf80/examination/

Course Summary

Color Perception - HVS

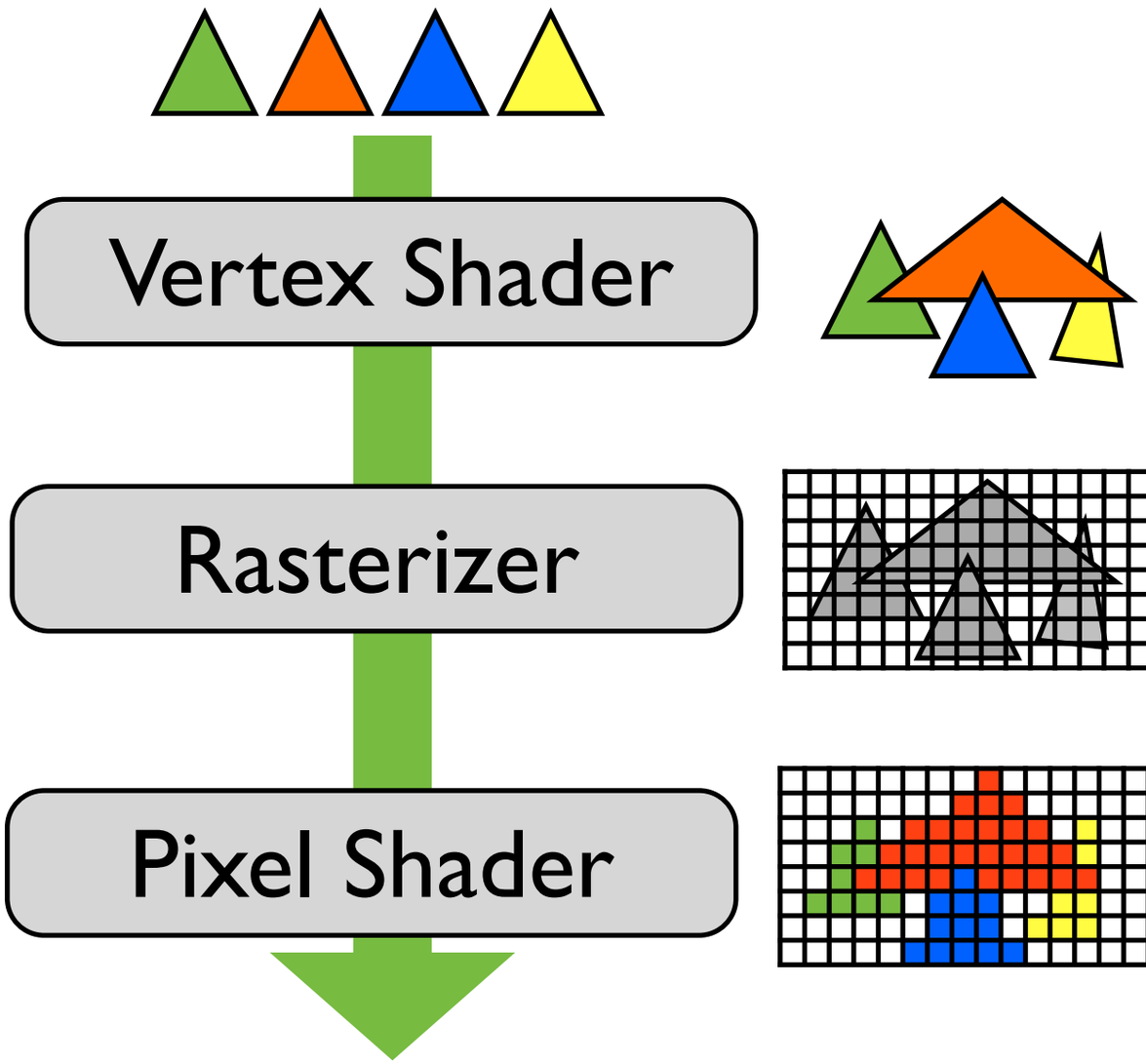
- Color stimulates cones in the retina
  - **Rods**: monochromatic, night vision
  - **Cones**: Color sensitive, three types of cones





Graphics Hardware

- **Pipeline** that accelerates the costly tasks of rendering



Move geometry

Compute visibility per pixel

Compute color per pixel

Real-time vs Offline

- Real-time
  - Render image in ~20 ms
  - Instant feedback
  - User interactions
- Offline (feature films)
  - Each image may take hours or days
  - Photorealism
  - No user interaction

Shading

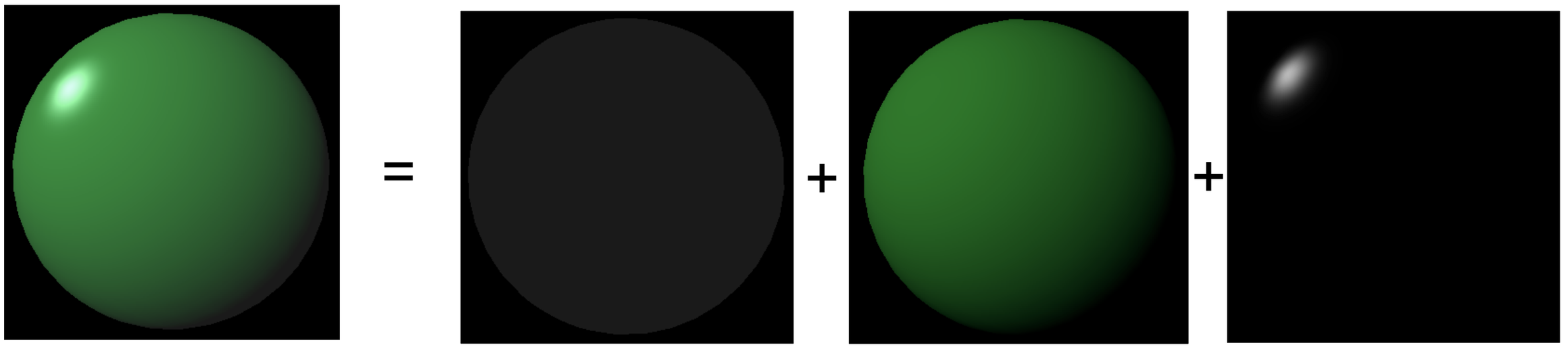
- Flat, Gouraud & Phong Shading
- Texture Mapping
- Bump Mapping
- Reflection Mapping
- Procedural Shaders (concept)

Phong Model

- Combine the three terms

$$I = k_a L_a + k_d L_d \max(\mathbf{l} \cdot \mathbf{n}, 0) + k_s L_s \max((\mathbf{r} \cdot \mathbf{v})^\alpha, 0)$$

Ambient Diffuse Specular

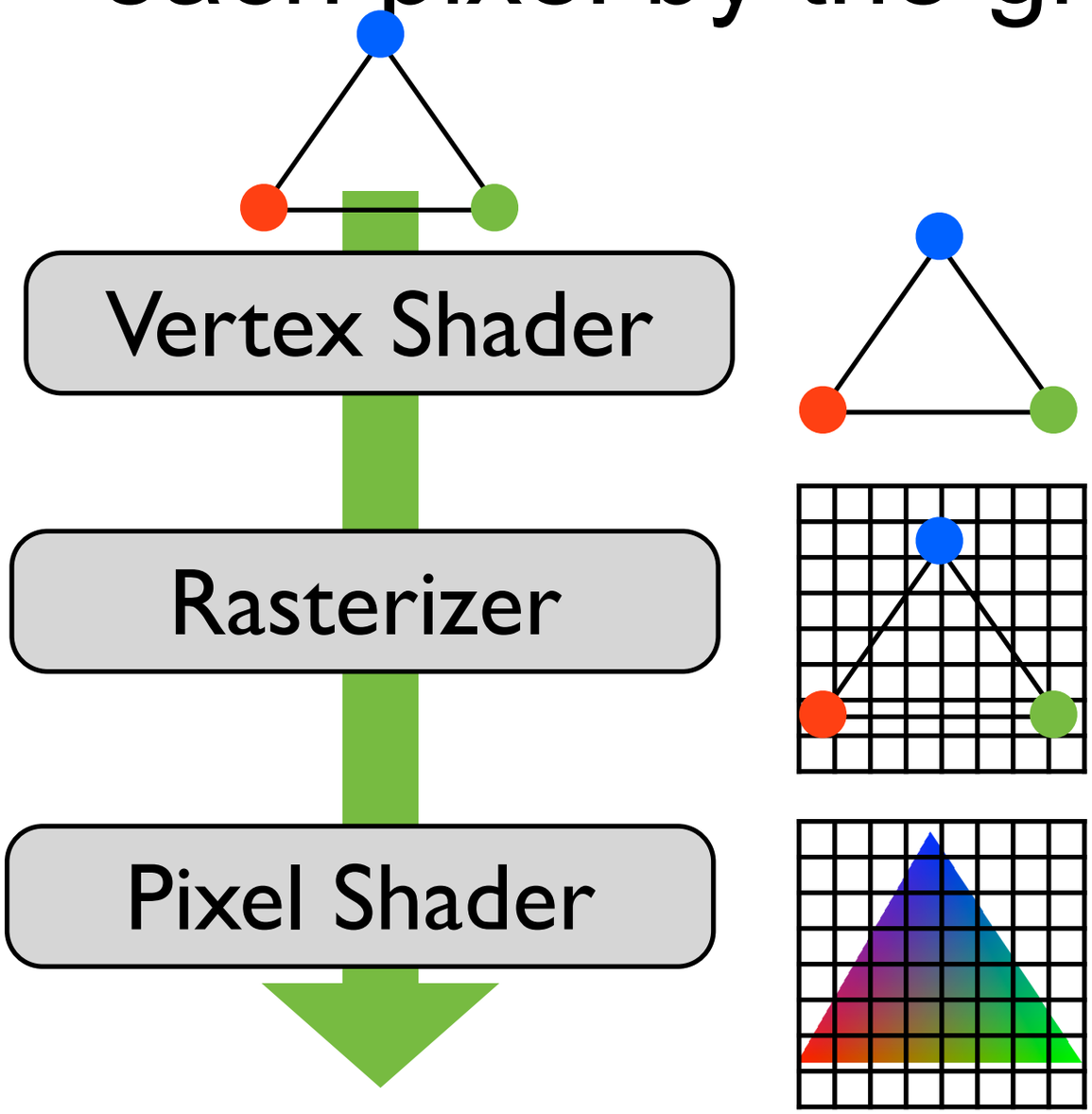


GLSL

- How to transform a point and vector
- How to implement a Phong shader
- How to pass information to a shader, and vertex data from VS to PS
  - Constants, which are the same for all invocations of the shader in a frame: `uniform`
  - Information that varies per vertex: `in, out`
- Difference between VS and PS

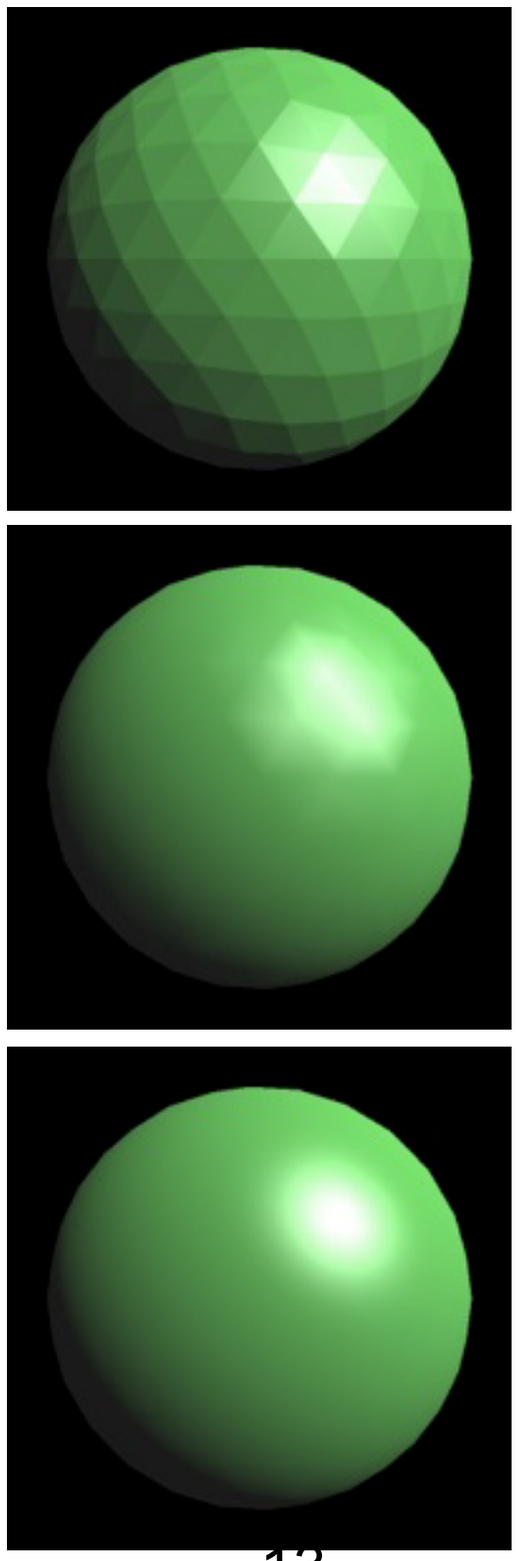
Hardware Interpolation

- Vertex attributes are interpolated for each pixel by the graphics hardware



Polygonal Shading

- Flat Shading
  - Set **l**, **n**, **v** constant for **each triangle**
  - Compute shading once per triangle
- Gouraud Shading
  - Shade at **each vertex**
  - Interpolate between the three vertex colors for each fragment within triangle
- Per-Pixel Phong Shading
  - Compute shading for **each pixel**



Parametric Surfaces

- Given function: $\mathbf{s}(u, v) = \ldots$

  - Define position, tangent, binormal and normal for any value of (u,v), i.e., tangent space.

  - Ex: plane, sphere

  - Describe how to go from $\mathbf{s}(u, v) = \ldots$ to a triangular mesh

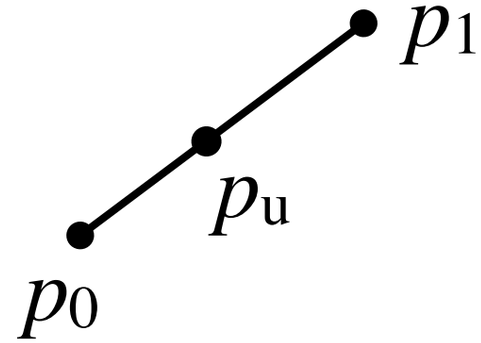
Interpolation

- Linear Interpolation
- Bilinear interpolation
- *Barycentric interpolation*
- Hermite Interpolation
  - Two points + two tangents define curve segment
  - Smoothstep
- Catmull-Rom Interpolation
  - Four points define curve segment

Bi-linear interpolation

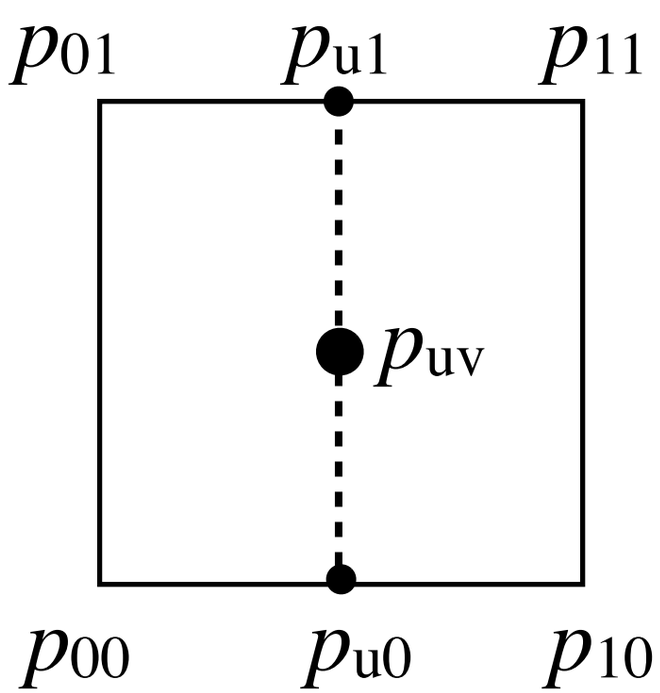
- Remember linear interpolation

$$p_u = (1-u)p_0 + up_1$$



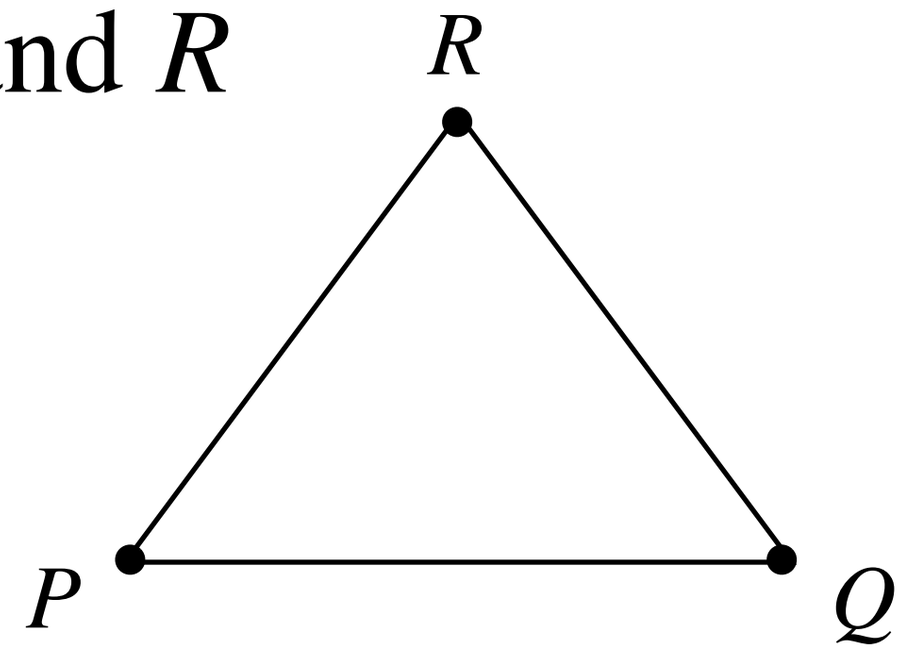
- Bilinear interpolation:

$$p_{u0} = (1-u)p_{00} + up_{10}$$
$$p_{u1} = (1-u)p_{01} + up_{11}$$
$$p_{uv} = (1-v)p_{u0} + vp_{u1}$$



Triangle

- Defined by three points P, Q and R



- Points inside triangle:

$$wP + uQ + vR$$

- u,v,w : barycentric coordinates

$$u + v + w = 1$$

$$u, v, w \geq 0$$

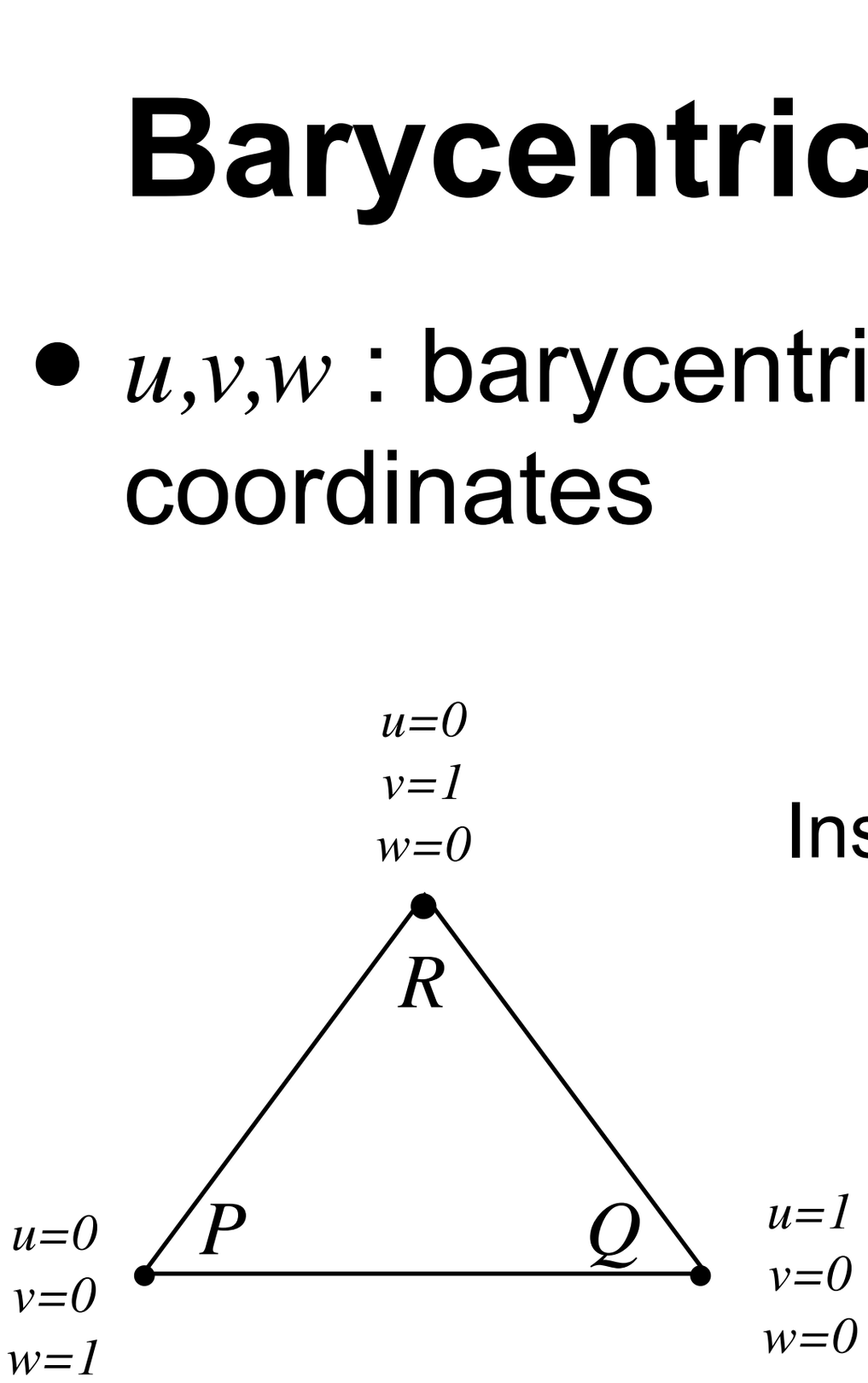
Barycentric Interpolation

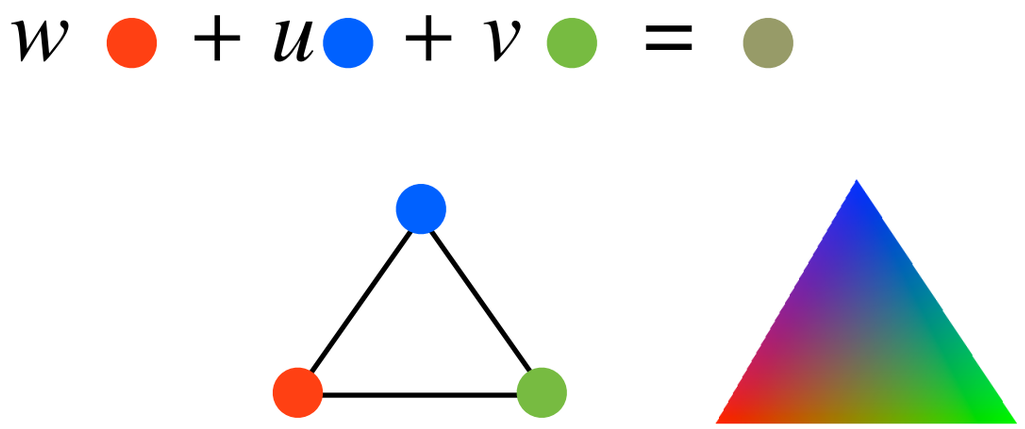
- u,v,w : barycentric coordinates

$$wP + uQ + vR$$

$$u + v + w = 1$$

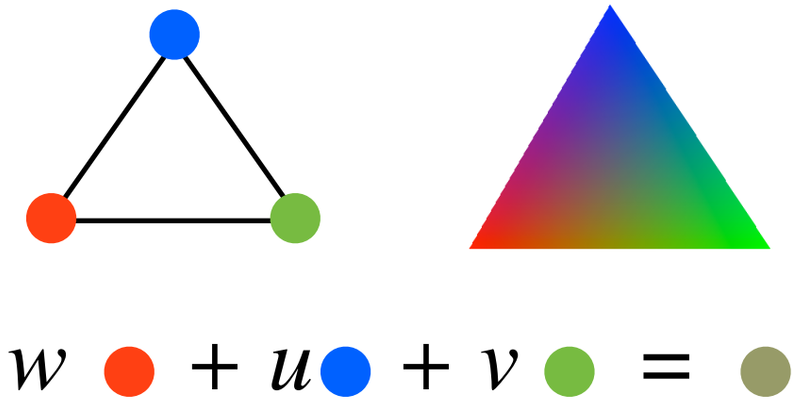
Inside triangle if: $u, v, w \geq 0$

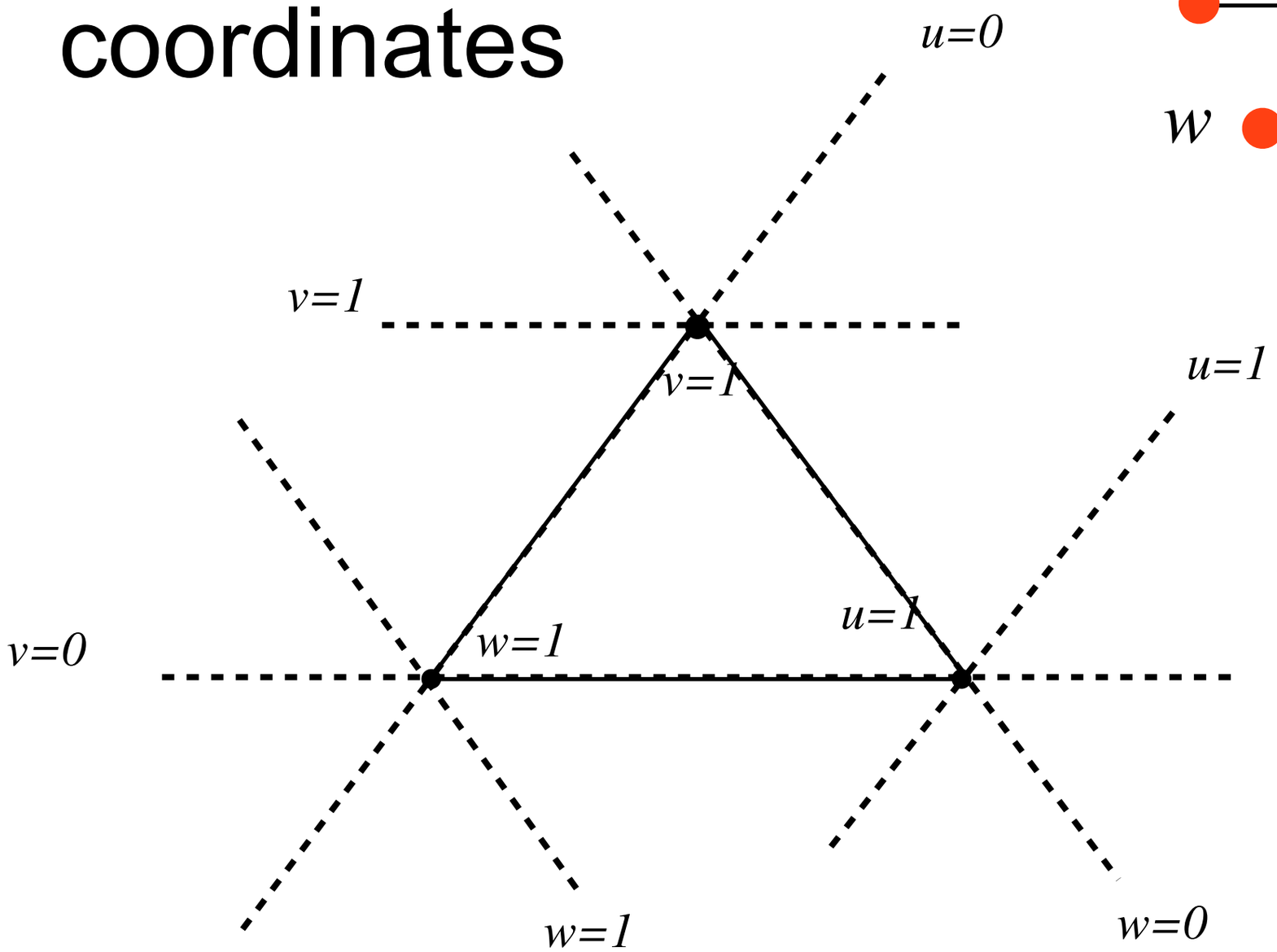




Barycentric Interpolation

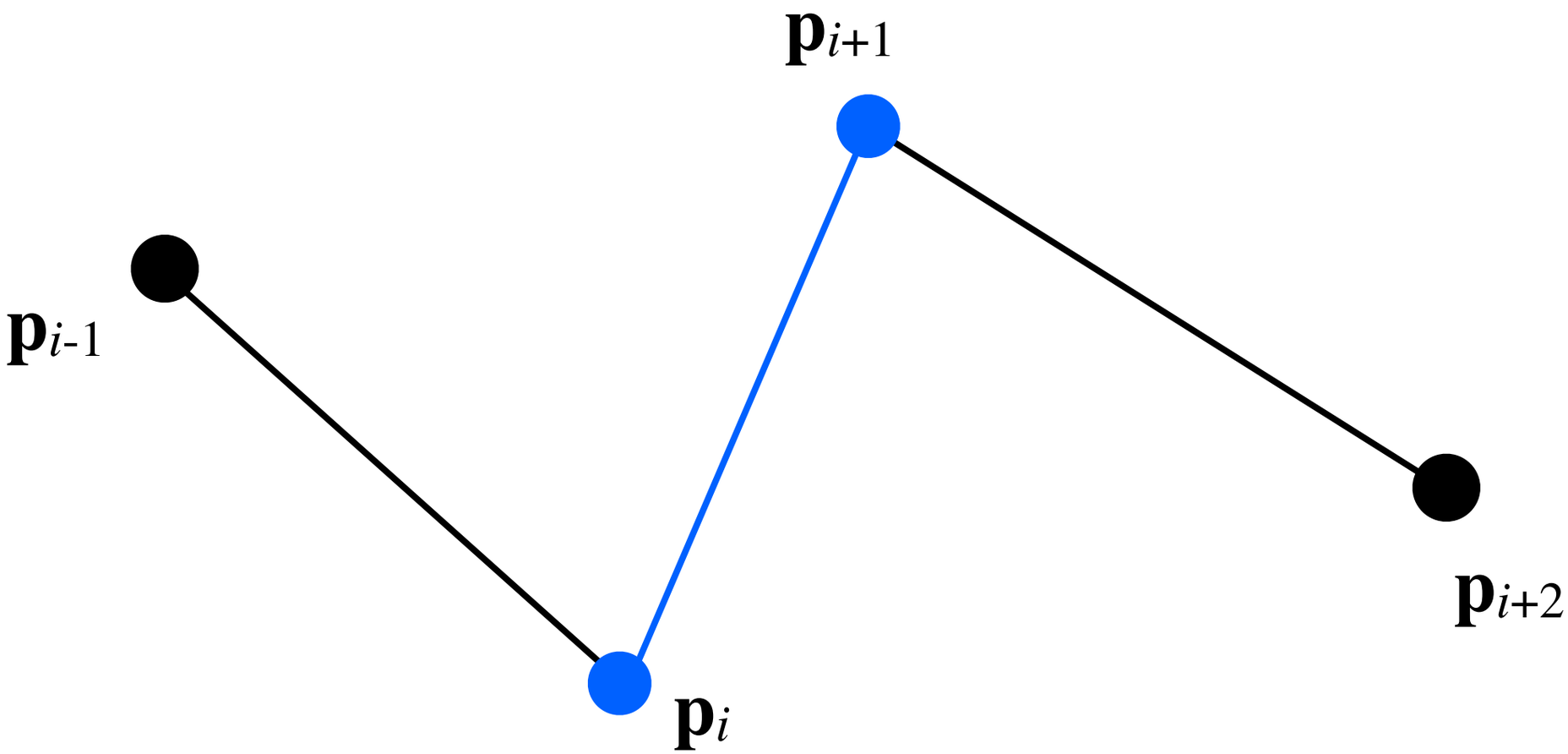
- u,v,w : barycentric coordinates



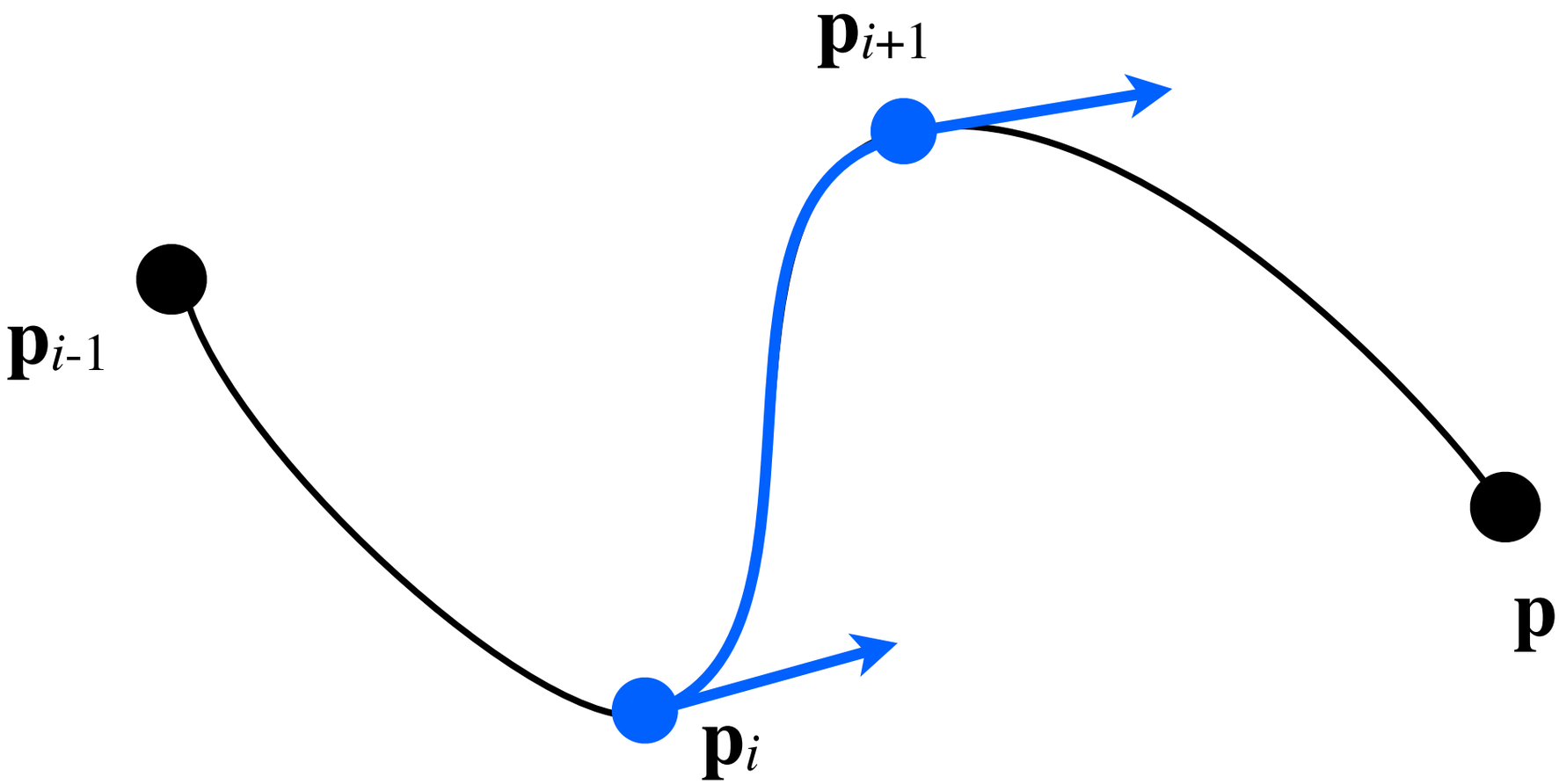


$$u + v + w = 1$$

Linear interpolation

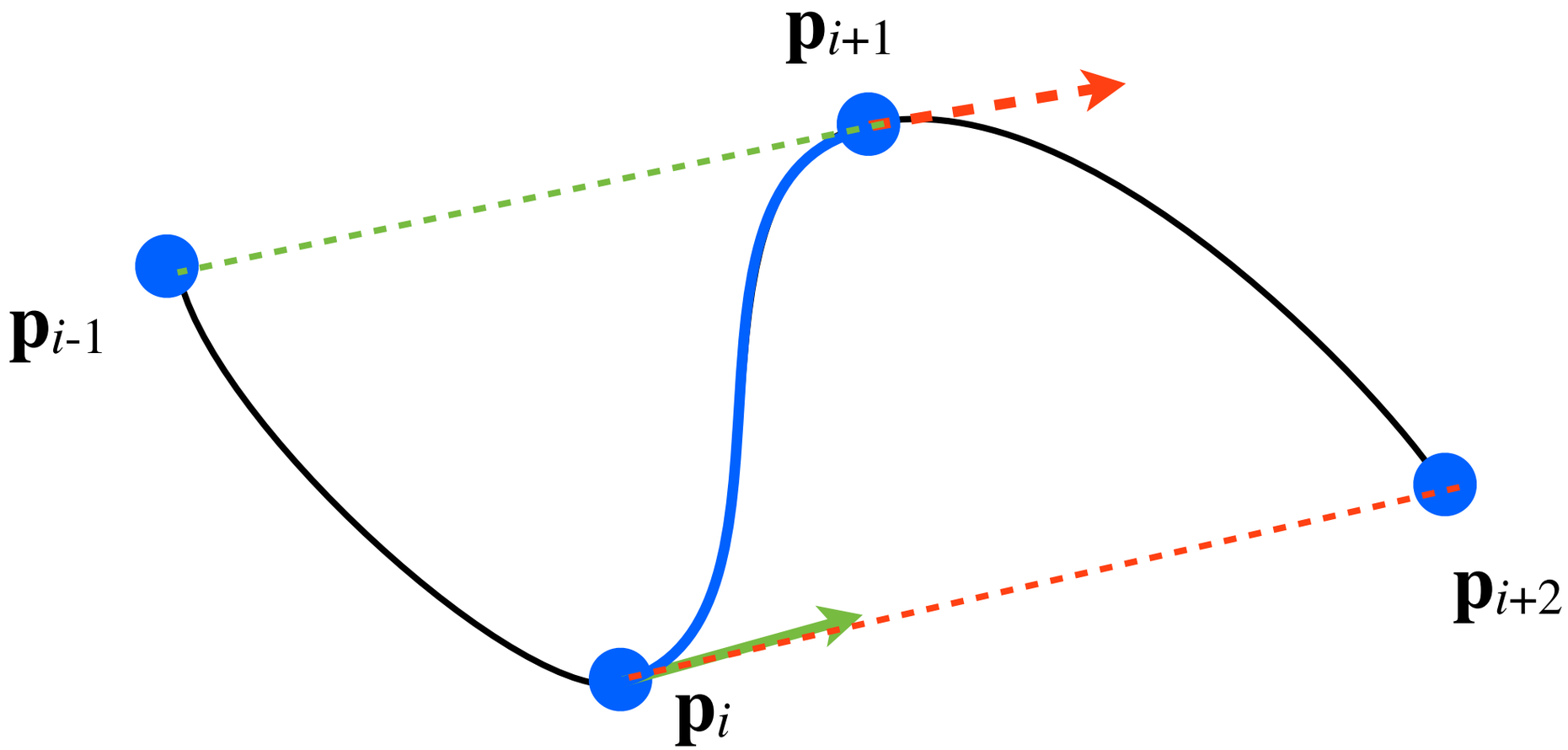


Hermite Interpolation



Use two points and two tangents
to specify **cubic** curve segment

Catmull-Rom Interpolation



Use **four points** to specify **cubic** curve segment. No need to manually specify tangents. Tangents approximated by: $\mathbf{p}'_i \approx \frac{\mathbf{p}_{i+1} - \mathbf{p}_{i-1}}{2}$

Classification of Transforms

Translation

Rotation

Rigid Body
preserves angles
and distances

Uniform Scaling

Similarity
preserves angles

Non-Uniform Scaling

Shear

Reflection

Affine
preserves
parallel lines

Perspective

Projective preserves lines

Rotation Matrices

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

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

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

Rotation matrices are orthonormal

$$\mathbf{R}^{-1}(\theta) = \mathbf{R}^T(\theta) = \mathbf{R}(-\theta)$$

$$\mathbf{R}\mathbf{R}^t = 1$$

Memorize these for exam!

Transforms

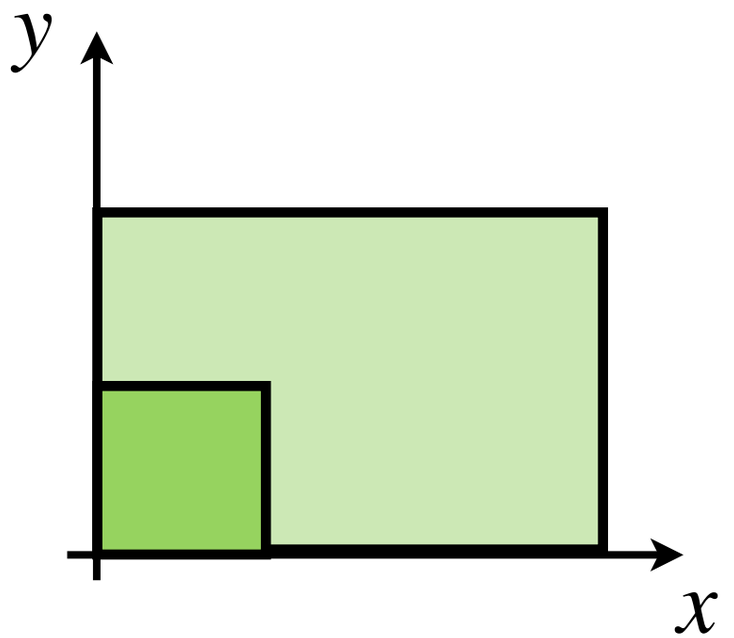
- Translation, scaling
- Rotation matrices
  - Remember rotation around X, Y and Z axes
  - Rotation around a point
  - Rotation around an arbitrary axis
- Concatenation of transform matrices

$$\mathbf{AB} \neq \mathbf{BA}$$

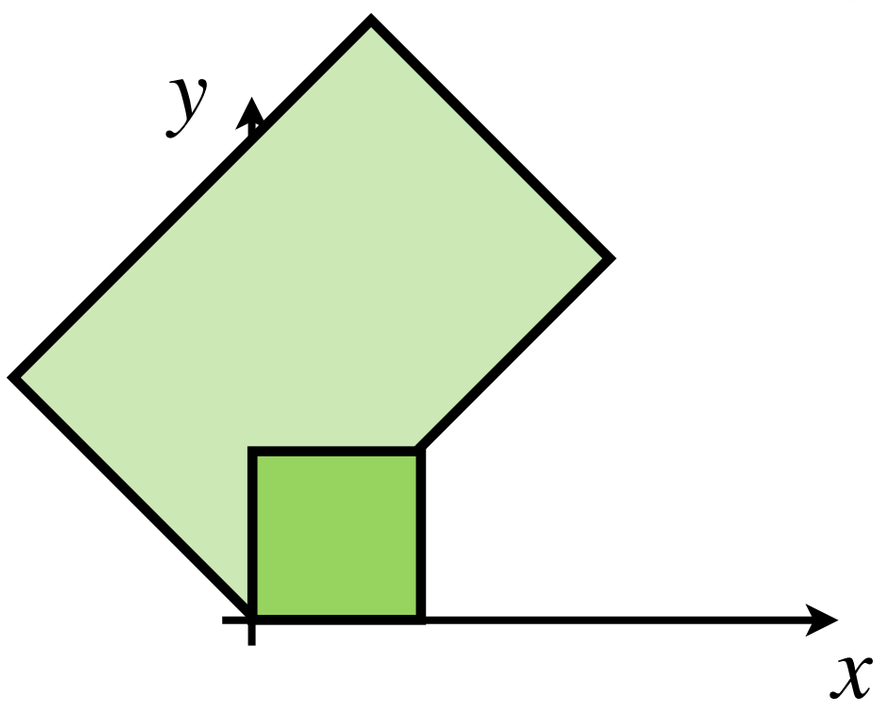
$$(\mathbf{ABC})^{-1} = \mathbf{C}^{-1}\mathbf{B}^{-1}\mathbf{A}^{-1}$$

Composite Transform

- First scale x by 3 and y by 2

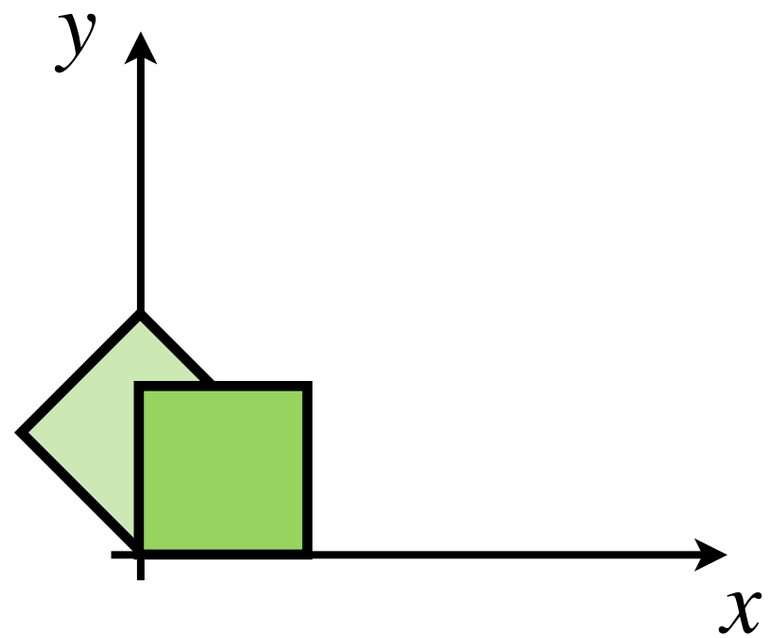


... then rotate $\pi/4$ degrees around z-axis

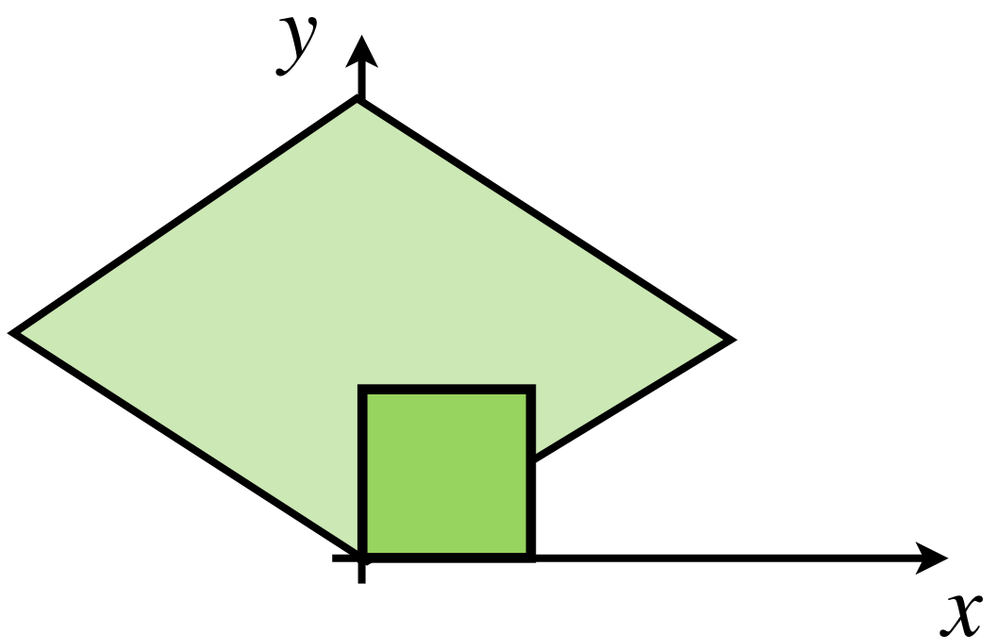


Composite Transform II

- Now: **Swap order** of transforms
- First rotate $\pi/4$ degrees around z-axis

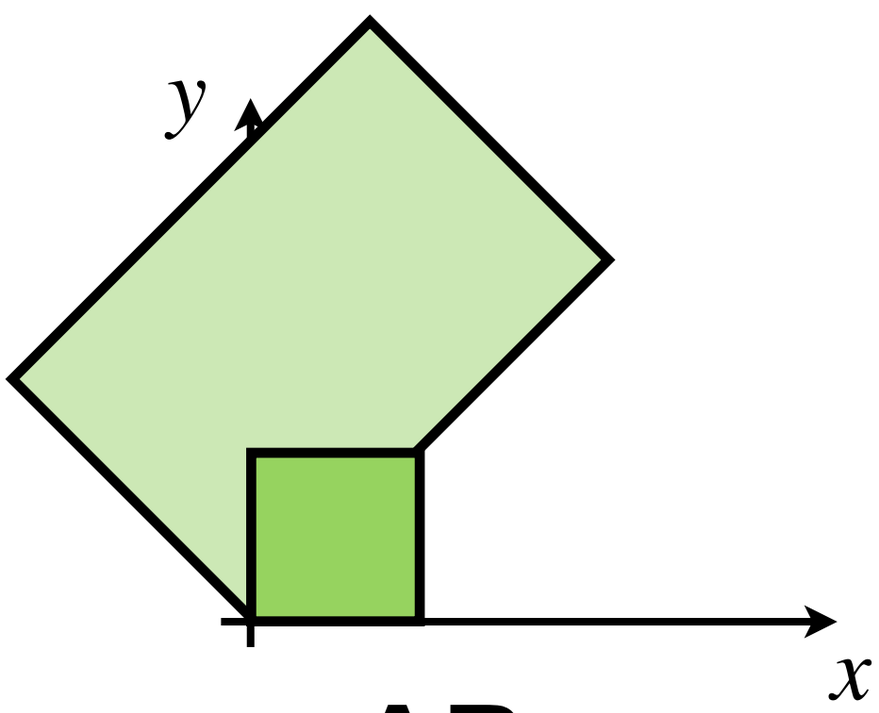


... then scale x by 3 and y by 2

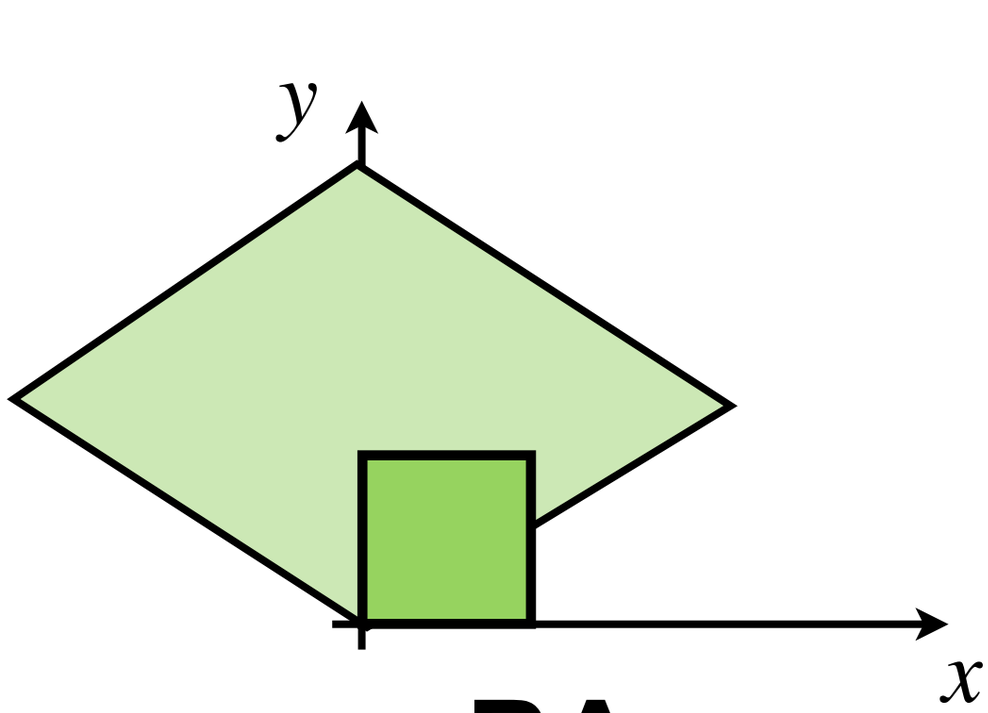


Composite transforms

- The concatenation of transform matrices is a new matrix with the same dimensions
- Order matters. In general: **AB ≠ BA**



AB

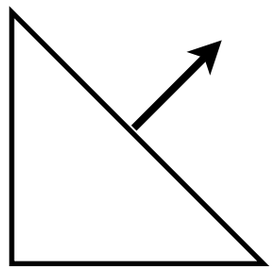


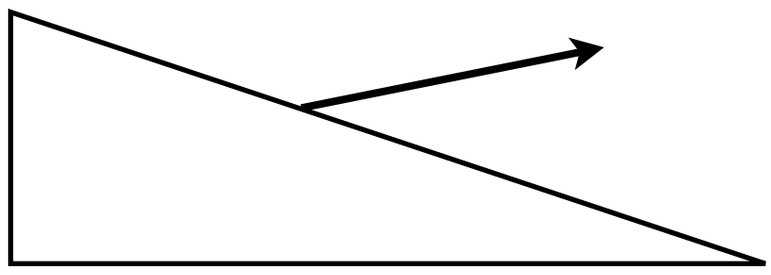
BA

Transforming Normals

- Can't transform normals as other vectors and points
  - Example: Non-uniform scaling $\mathbf{v}' = \begin{bmatrix} 3 & 0 \\ 0 & 1 \end{bmatrix} \mathbf{v}$

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



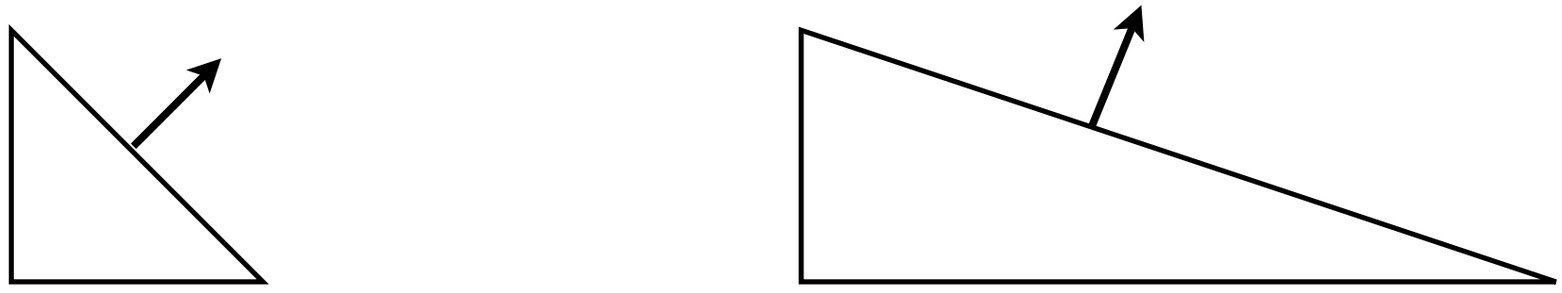


Use $(\mathbf{M}^{-1})^T$ to transform normals!

Transforming Normals

$$\mathbf{M} = \begin{bmatrix} 3 & 0 \\ 0 & 1 \end{bmatrix} \qquad \mathbf{M}^{-1} = (\mathbf{M}^{-1})^T = \begin{bmatrix} \frac{1}{3} & 0 \\ 0 & 1 \end{bmatrix}$$

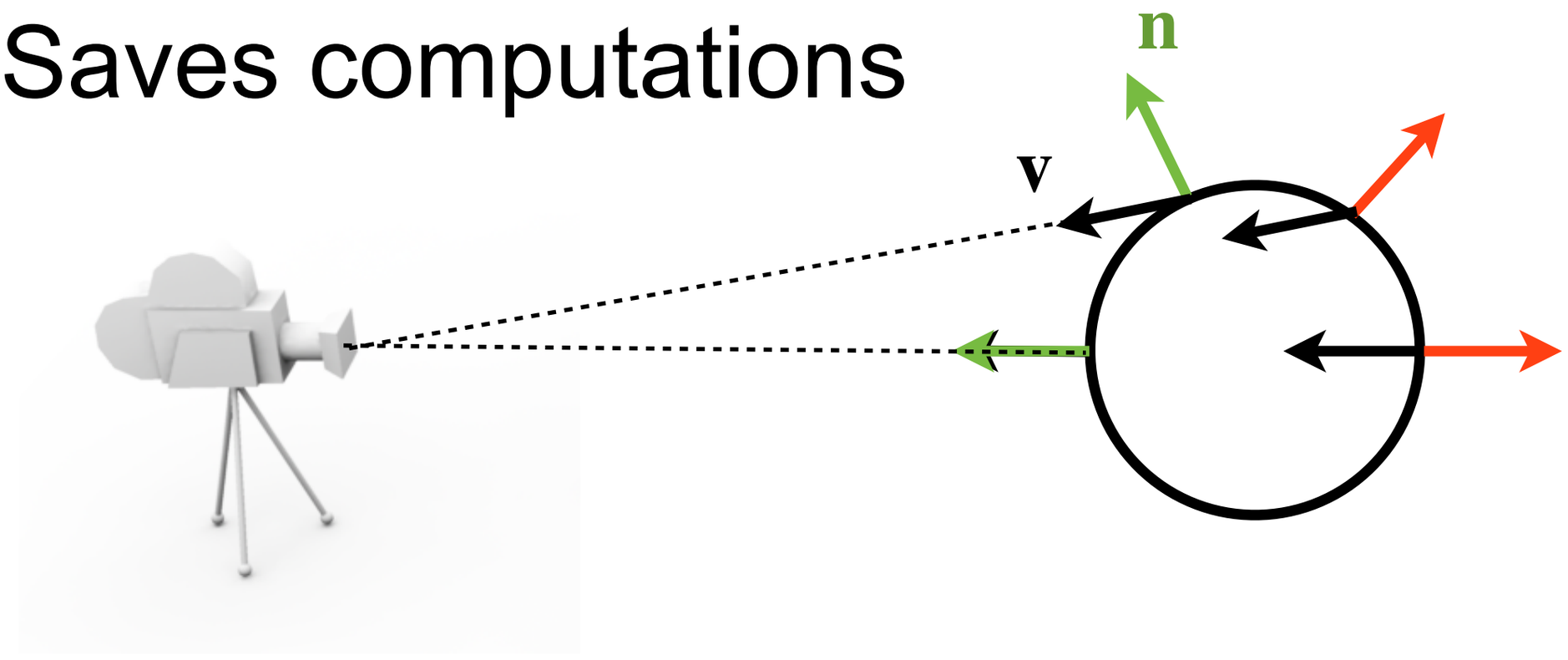
$$\mathbf{v}' = \begin{bmatrix} 3 & 0 \\ 0 & 1 \end{bmatrix} \mathbf{v} \qquad \mathbf{n}' = \begin{bmatrix} \frac{1}{3} & 0 \\ 0 & 1 \end{bmatrix} \mathbf{n}$$



Use $(\mathrm{M}^{-1})^T$ to transform normals!

Backface Culling

- Remove triangles facing away from the camera
- Saves computations



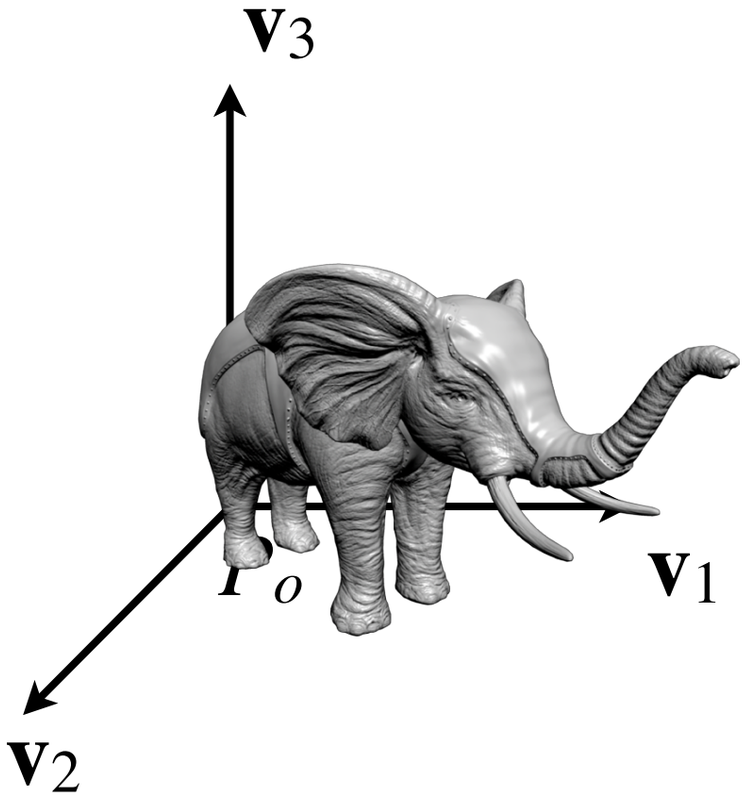
Backfacing if $\mathbf{n} \cdot \mathbf{v} < 0$

Coordinate Systems

- Construct an orthonormal coordinate system, change coordinate system
- Common coordinate systems:
  - Object space (Model space)
  - World space
  - View space (Camera space, Eye space)
  - Tangent space (useful for bump mapping)
  - Clip space (after multiplication with MVP)
  - NDC (after division by w)

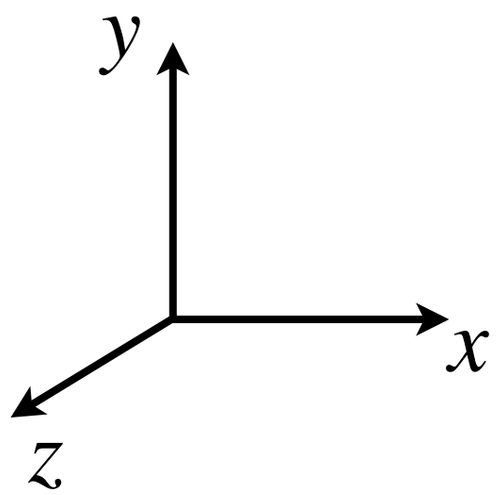
Object coordinates

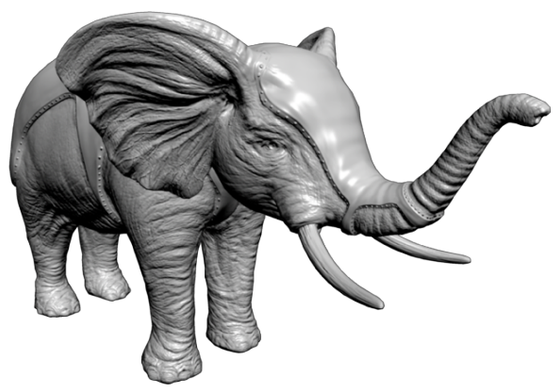
- The local coordinate system the model is defined in
- Often called “Object space”, “Model space” or “Model coordinates”



World coordinates

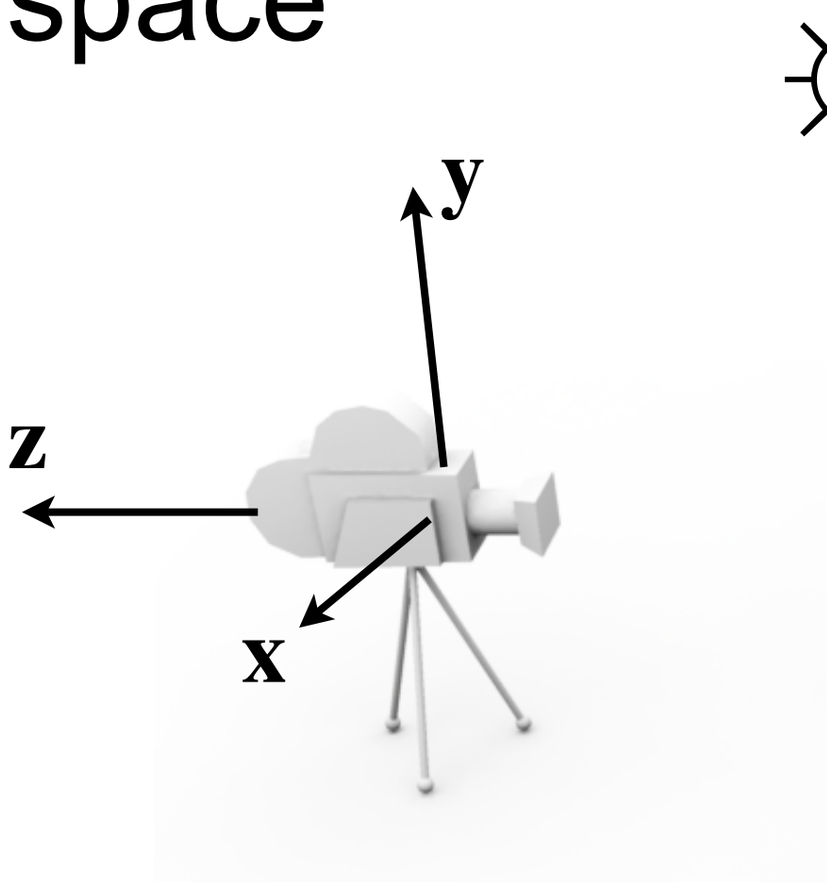
- Reference system in which all objects, lights and the camera are positioned
- Often referred to as “World space”

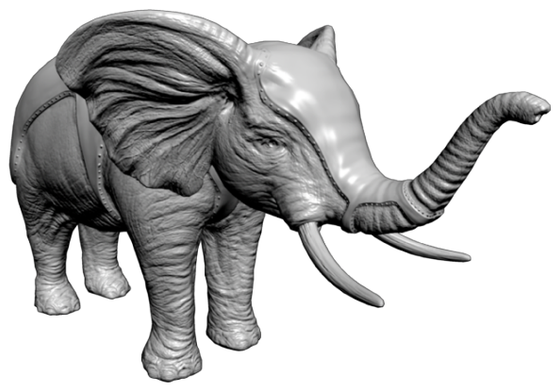




Eye (view) coordinates

- A coordinate system with the camera at center, looking along negative z
- "Camera space"

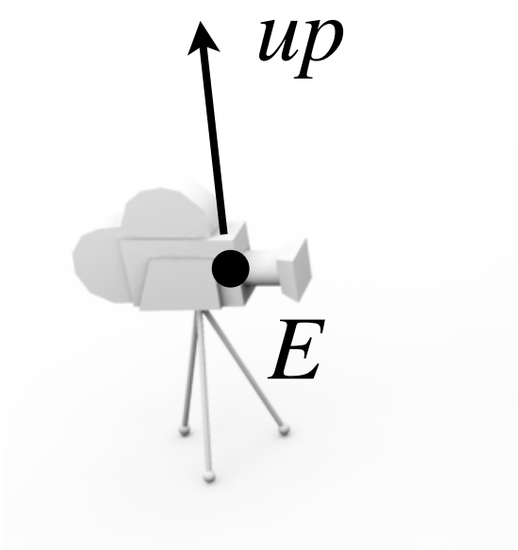




Setup Camera Matrix

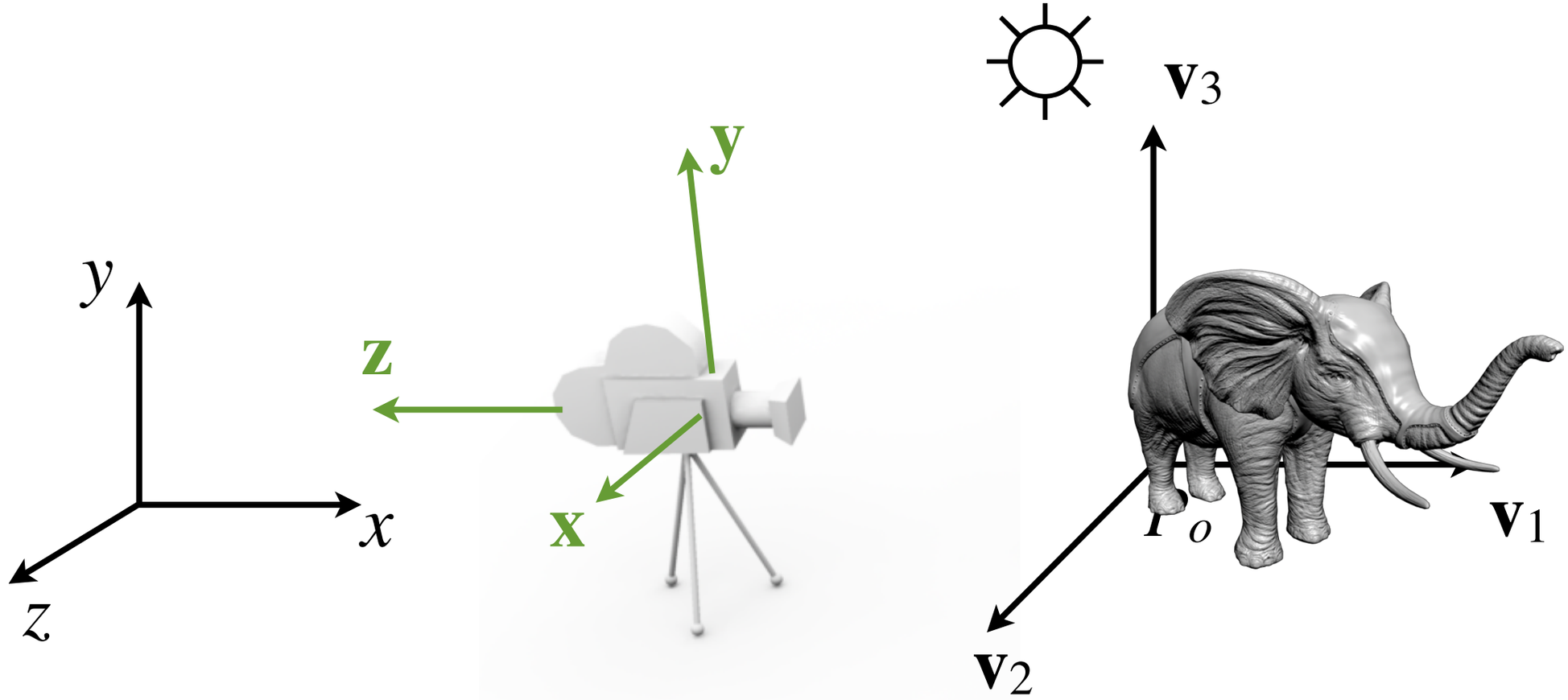
- LookAt function
  - Takes eye position (E), a position to look at (C) and an up vector (up)
  - Constructs the **View** matrix, i.e., a matrix that transforms geometry (in world space) into the camera's coordinate system (camera space)

```
mat4 View = LookAt(E.x,E.y,E.z,          // Camera position
                   C.x,C.y,C.z,          // Center of interest
                 up.x, up.y, up.z);  // Up-vector
```



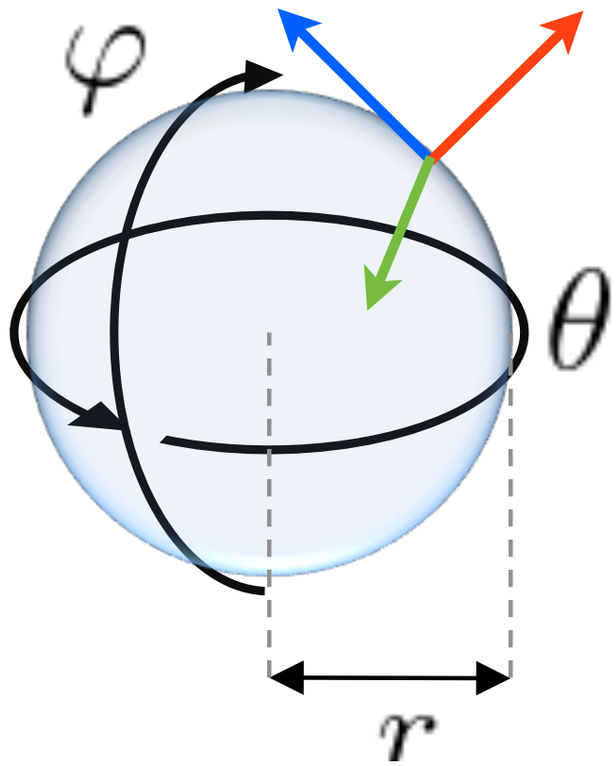
Change coordinate system

- Model or World matrix: From model to world
- View matrix: From world to view
- ModelView matrix: From model to view



Tangent space

- Local coordinate system made up of the tangent, binormal and normal
- Unique for each point of the surface
- When is tangent space useful?



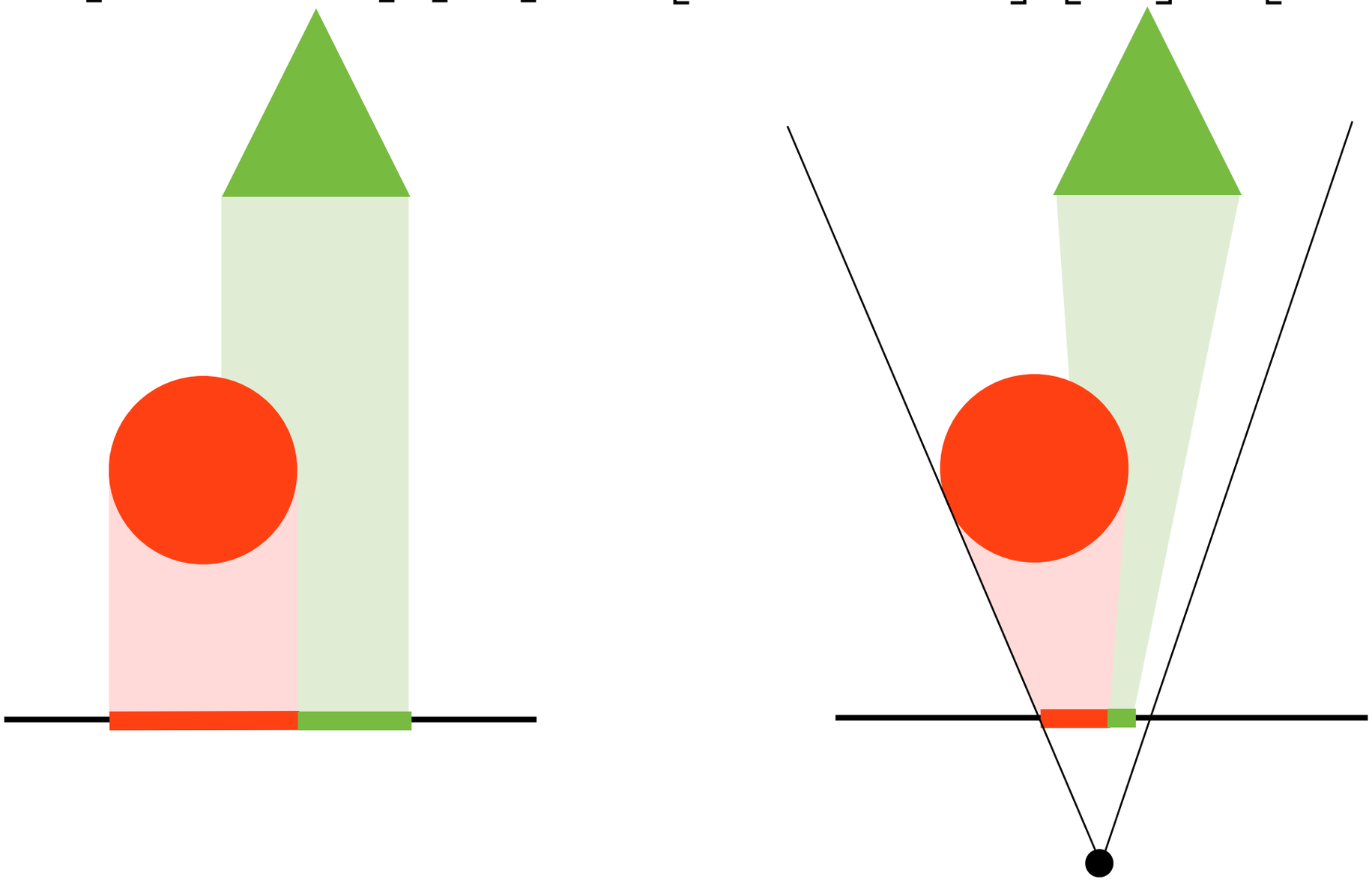
Projection

- From 3D to 2D
- Orthographic vs Perspective projection
- Perspective divide
- What does the ModelViewProjection Matrix do?

Orthographic vs Perspective

$$\begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

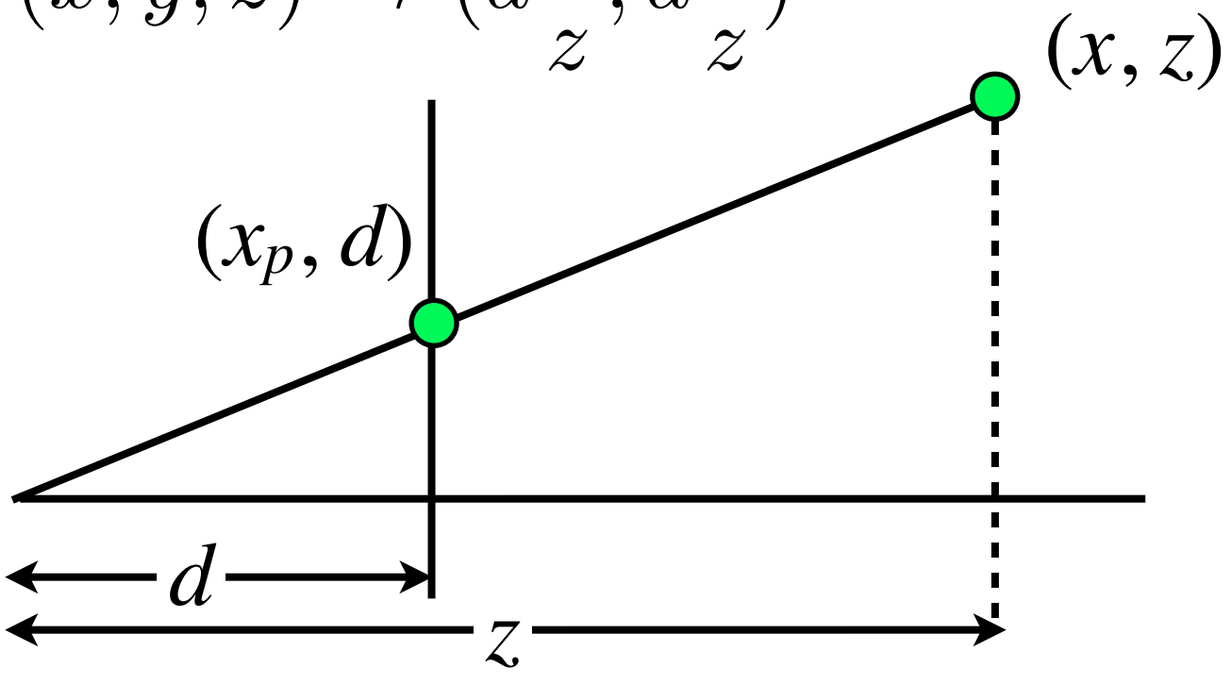
$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ -z \end{bmatrix} \rightarrow \begin{bmatrix} -\frac{x}{z} \\ -\frac{y}{z} \\ -1 \\ 1 \end{bmatrix}$$



Perspective Projection

- More realistic model - objects far away are smaller after projection

$$(x, y, z) \rightarrow (d\frac{x}{z}, d\frac{y}{z})$$

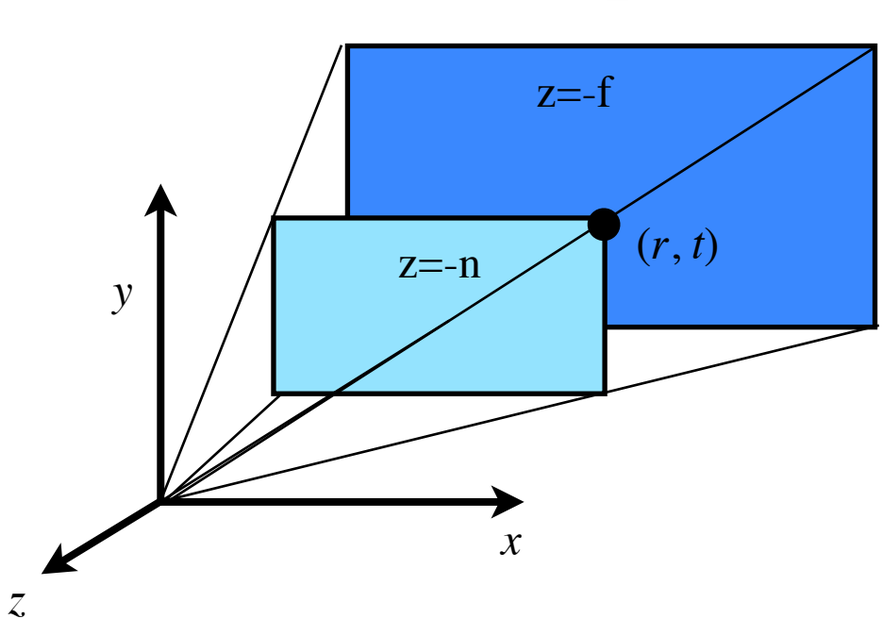


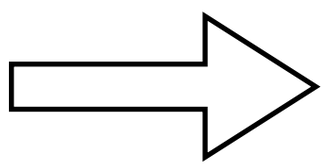
Equal triangles

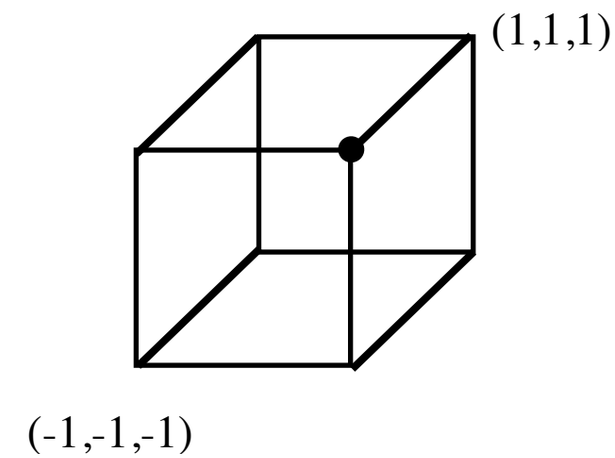
$$\frac{x_p}{d} = \frac{x}{z}$$

$$x_p = d\frac{x}{z}$$

OpenGL Projection







divide by w

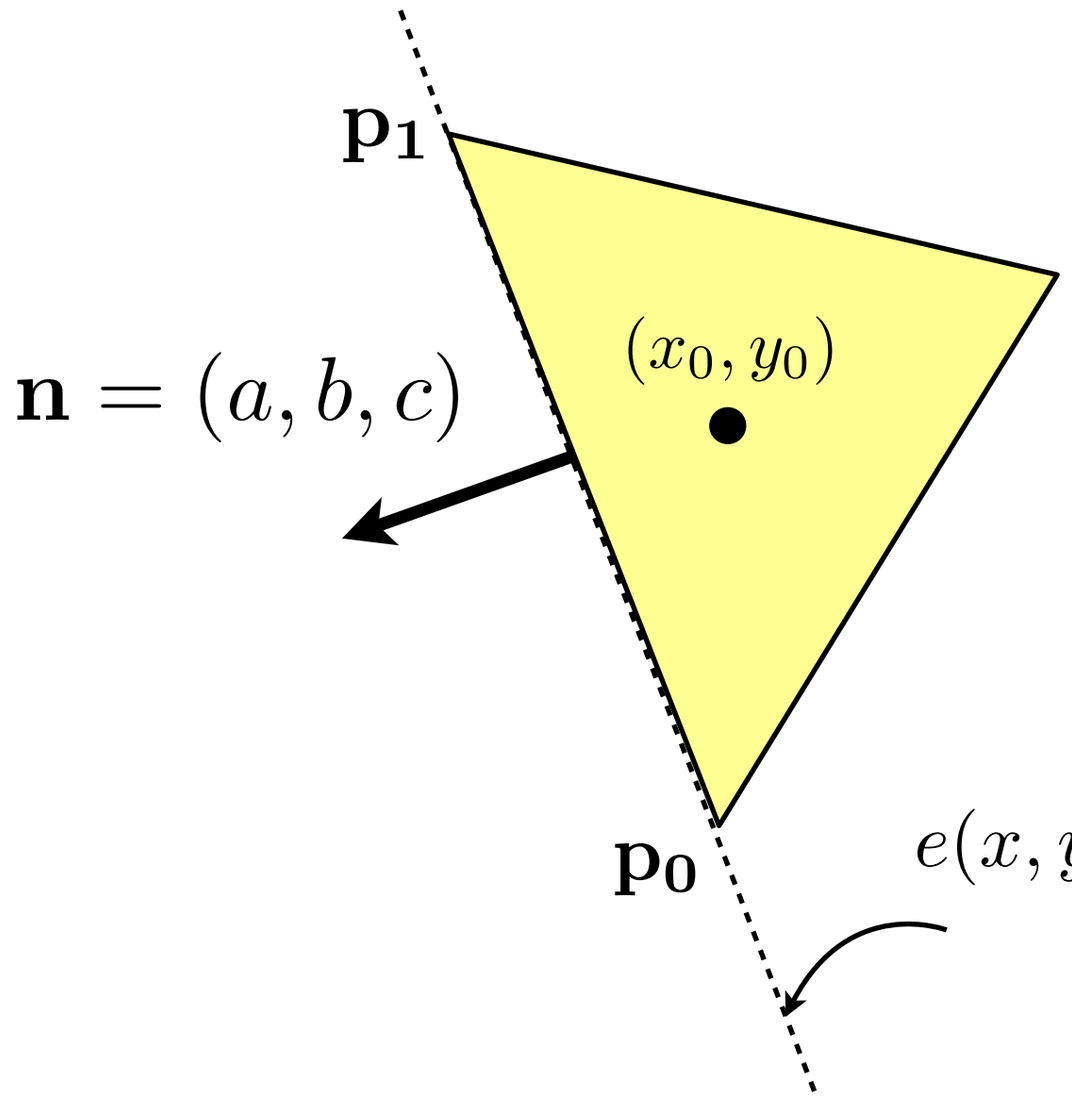
$$\begin{bmatrix} \frac{n}{r} & 0 & 0 & 0 \\ 0 & \frac{n}{t} & 0 & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x_c \\ y_c \\ z_c \\ w_c \end{bmatrix} \rightarrow \begin{bmatrix} \frac{x_c}{w_c} \\ \frac{y_c}{w_c} \\ \frac{z_c}{w_c} \\ 1 \end{bmatrix}$$

Projection Matrix | Camera space | Clip space | NDC

Normalized Device Coords

Edge Equation

- Point inside edge test



Point (x_0, y_0) inside edge between p_0 and p_1 if:

$$\mathbf{n} \cdot (x_0, y_0, 1) < 0$$

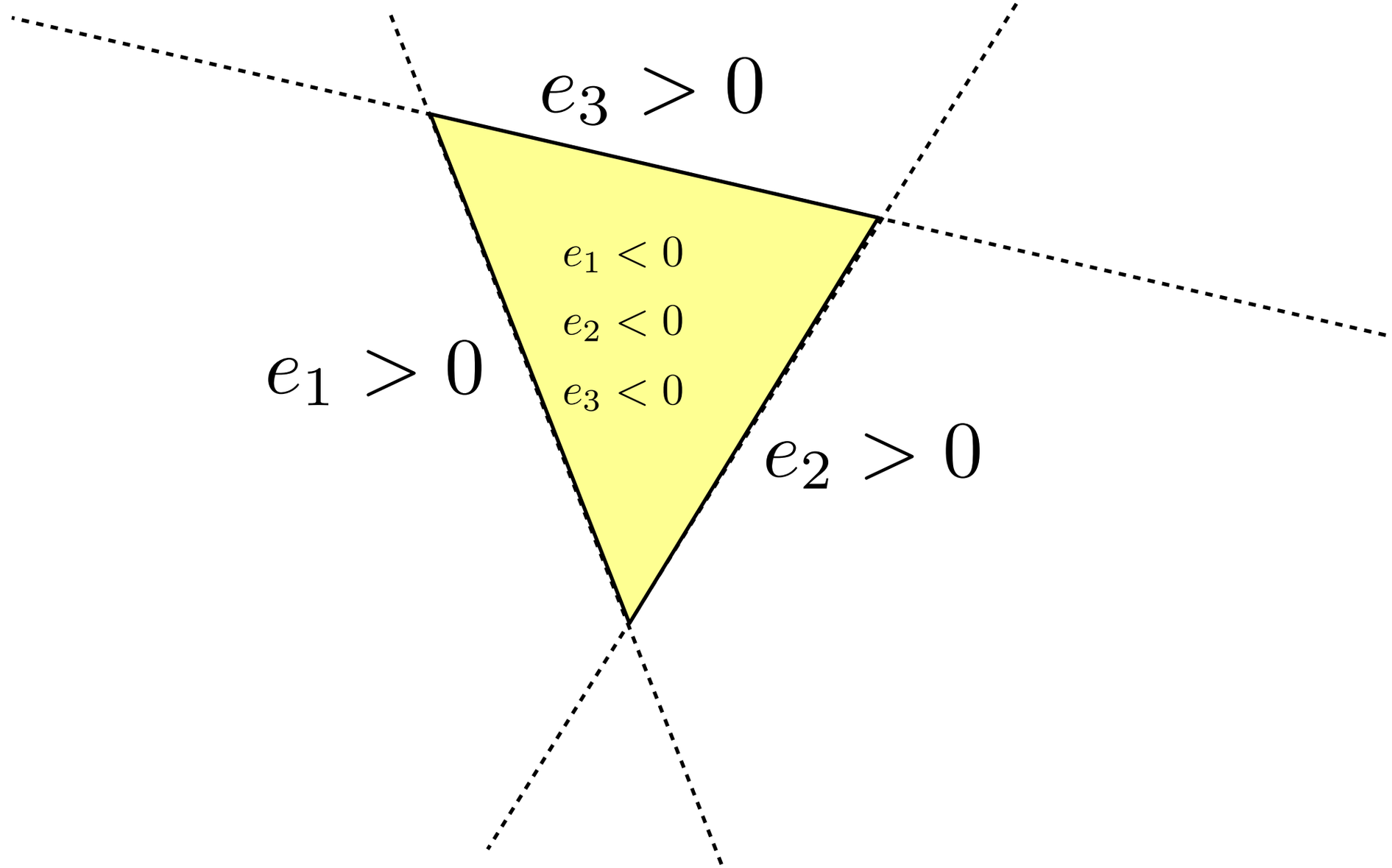
$$ax_0 + by_0 + c < 0$$

$$e(x, y) = (\mathbf{p}_0 \times \mathbf{p}_1) \cdot (x, y, 1)$$

$$= ax + by + c$$

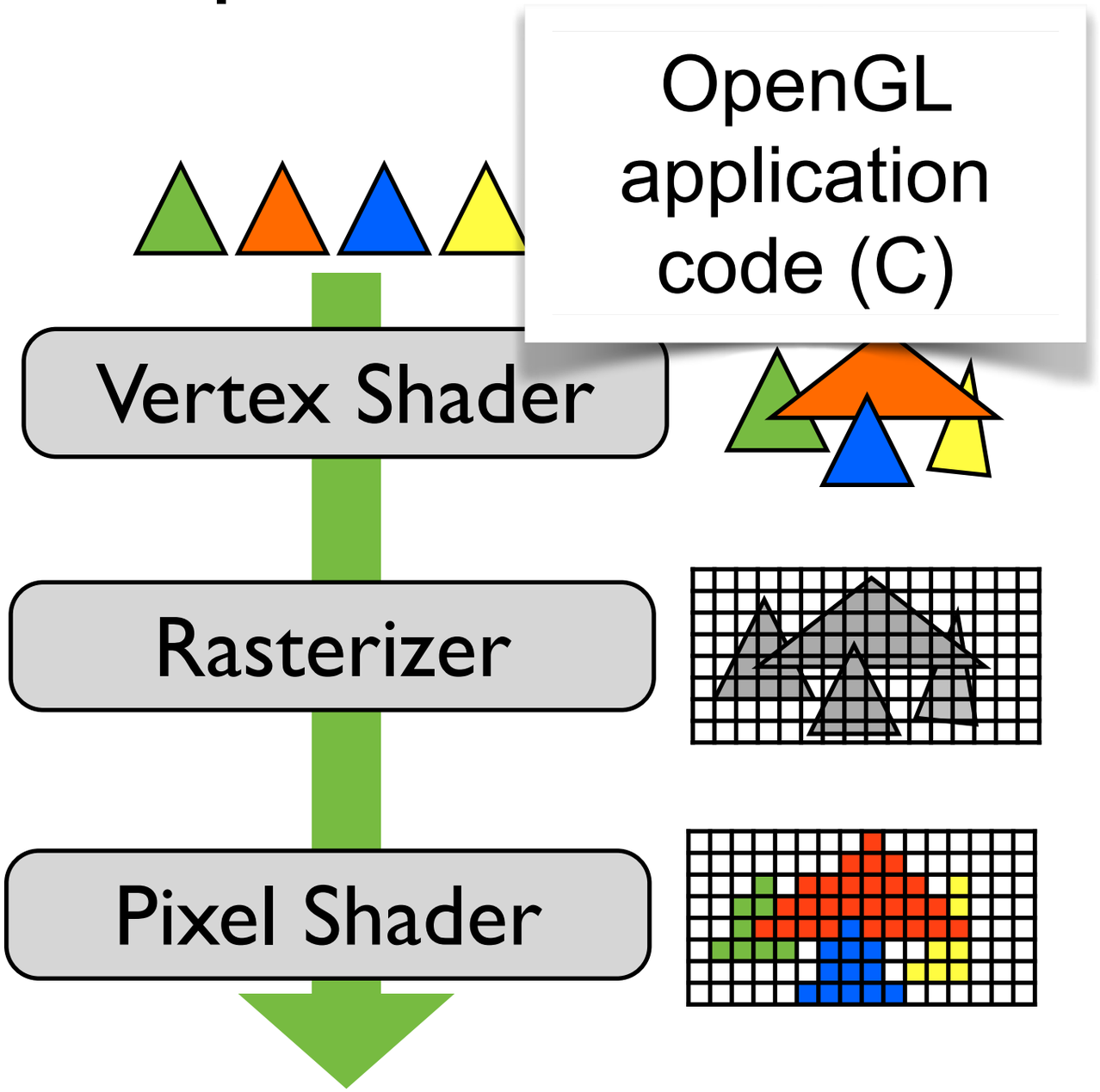
Point Inside Triangle Test

Inside all three edges: **Hit**



Graphics Hardware

- Expresses the rendering pipeline



Graphics Pipeline Summary

- Application setup
  - Geometry, shaders and transform matrices
- For each triangle:
  - Apply transforms (e.g., **MVP**) in **vertex shader**
  - Project on screen (divide by w coordinate)
  - Rasterize: Evaluate edge equations at pixel center
  - For each covered pixel:
    - Depth buffer test to check if closest object
    - If visible: run **pixel shader**, store in color buffer

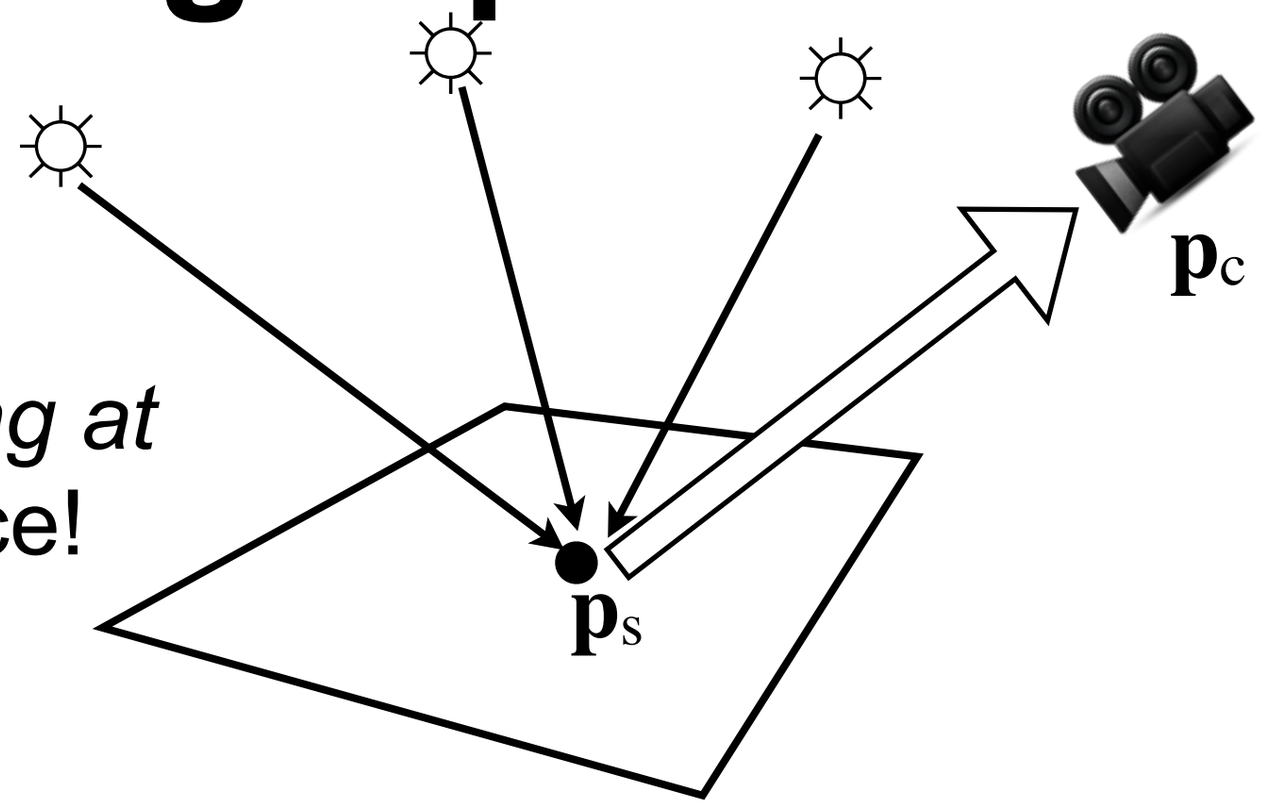
Depth Buffering

- For each pixel, store a depth value
  - Initialize to large value
- For each pixel, compute depth value d_{new}, of **current triangle** at hitpoint
  - If $d_{\text{new}} < d_{\text{stored}}$ we have a hit. Update the depth buffer: $d_{\text{stored}} := d_{\text{new}}$, and call the pixel shader
  - Otherwise, the triangle is covered by already drawn primitives. Move to next pixel.

Visibility Computations

- Rasterization pipeline
  - For **each triangle**, find **covered pixels**
  - Check with depth buffer if closest hit
- Ray tracing
  - For **each pixel**, find **overlapping triangles**
  - Shoot a ray from the camera through the pixel
- Loops are reversed!

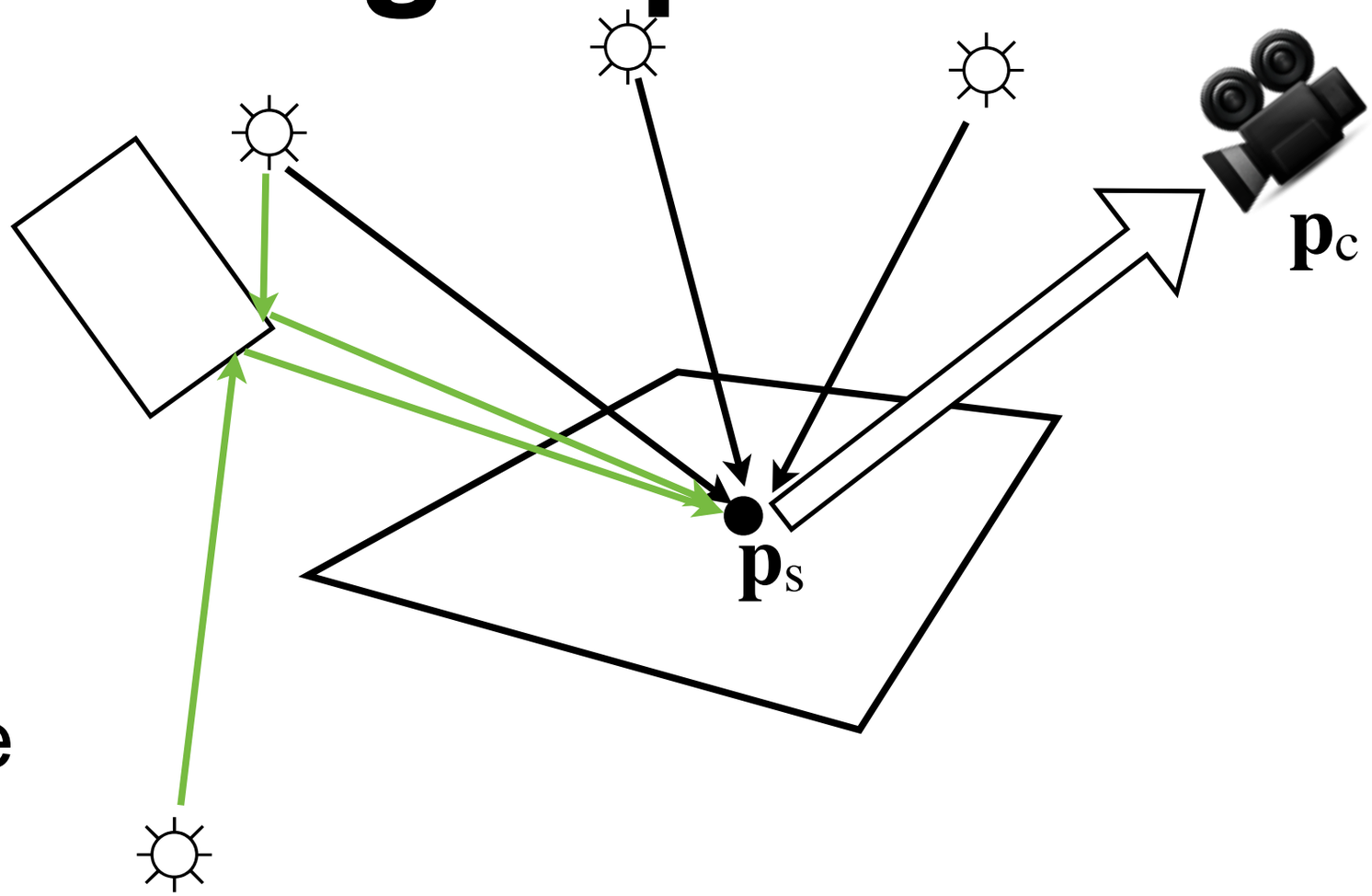
The Rendering Equation



Energy conservation:
Total amount of light *arriving at* and *leaving* $\mathbf{p}_S$ must balance!

The total intensity of light leaving point $\mathbf{p}_S$ is equal to the incoming light energy from all possible points plus the emission of light from $\mathbf{p}_S$ itself (if $\mathbf{p}_S$ is a light source)

The Rendering Equation



Also, light may have bounced several times before arriving at $\mathbf{p}_S$

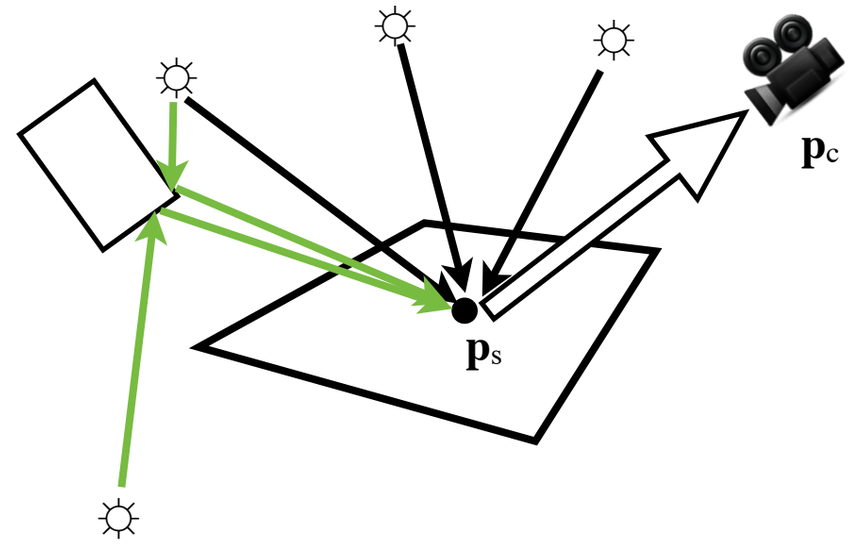
The Rendering Equation

$$I(\mathbf{p}_c, \mathbf{p}_s) = v(\mathbf{p}_c, \mathbf{p}_s)\left[\epsilon(\mathbf{p}_c, \mathbf{p}_s) + \int_S \rho(\mathbf{p}_c, \mathbf{p}_s, \mathbf{p}) I(\mathbf{p}_s, \mathbf{p}) d\mathbf{p}\right]$$

$I(\mathbf{p}_c, \mathbf{p}_s)$ Intensity of light from $\mathbf{p}_s$ towards $\mathbf{p}_c$

$v(\mathbf{p}_c, \mathbf{p}_s)$ Visibility between $\mathbf{p}_s$ and $\mathbf{p}_c$ (0 or 1)

$\epsilon(\mathbf{p}_c, \mathbf{p}_s)$ Self-emitted light from $\mathbf{p}_s$ toward $\mathbf{p}_c$



$\rho(\mathbf{p}_c, \mathbf{p}_s, \mathbf{p})$ BRDF: Fraction of light from $\mathbf{p}$ towards $\mathbf{p}_s$ scattered in the direction of $\mathbf{p}_c$.
The BRDF describes the material properties at $\mathbf{p}_s$

$\int_S \ldots d\mathbf{p}$ Integral over all possible points in the scene

Rendering Equation Summary

- The general equation is very complex. Why?

$$I(\mathbf{p}_c, \mathbf{p}_s) = v(\mathbf{p}_c, \mathbf{p}_s) \left[\epsilon(\mathbf{p}_c, \mathbf{p}_s) + \int_S \rho(\mathbf{p}_c, \mathbf{p}_s, \mathbf{p}) I(\mathbf{p}_s, \mathbf{p}) d\mathbf{p} \right]$$

- Simplifications (see Lecture 6) leads to plausible approximations, such as Phong:

$$I(\mathbf{p}_s, \mathbf{p}_c) \approx \sum_i \rho(\mathbf{v}, \mathbf{l}_i) L_i = \sum_i (k_a + k_d(\mathbf{l}_i \cdot \mathbf{n}) + k_s(\mathbf{r}_i \cdot \mathbf{v})^\alpha) L_i$$

Assignments

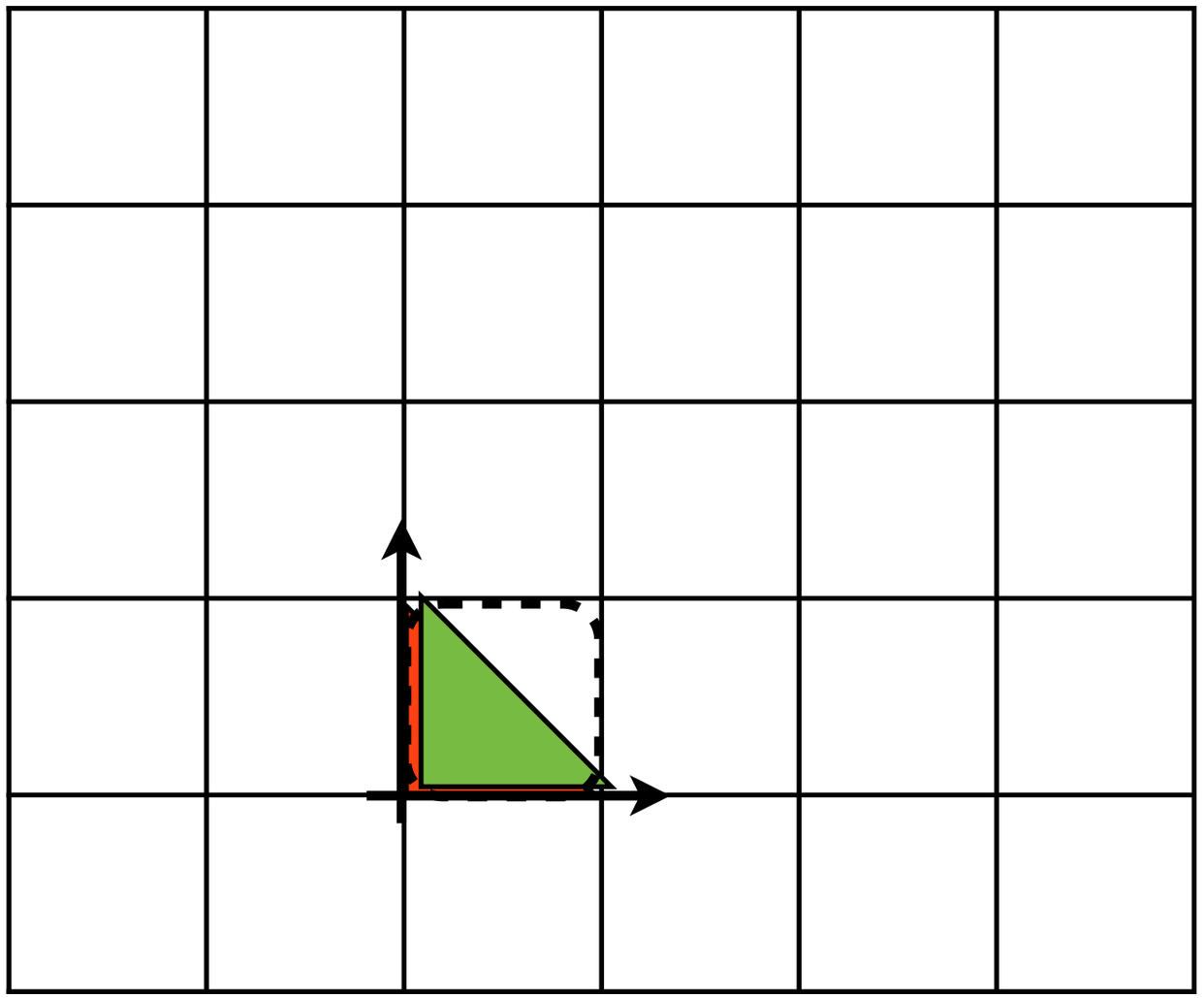
- The assignments are designed to cover many important course topics
- When preparing for the exam, revisit the assignments and seminars
  - Use lab 1 to investigate transform examples
  - Concatenations, hierarchical modeling
  - Shading - change shading parameters and get intuition for the visual look
  - Bump mapping - done twice

Simple Scene Graph

- Hierarchical Modeling
- Spin versus Orbit
- TRS transform for each node
  - Scaling is applied first, then rotation and finally translation.
- “What does this code fragment do?”

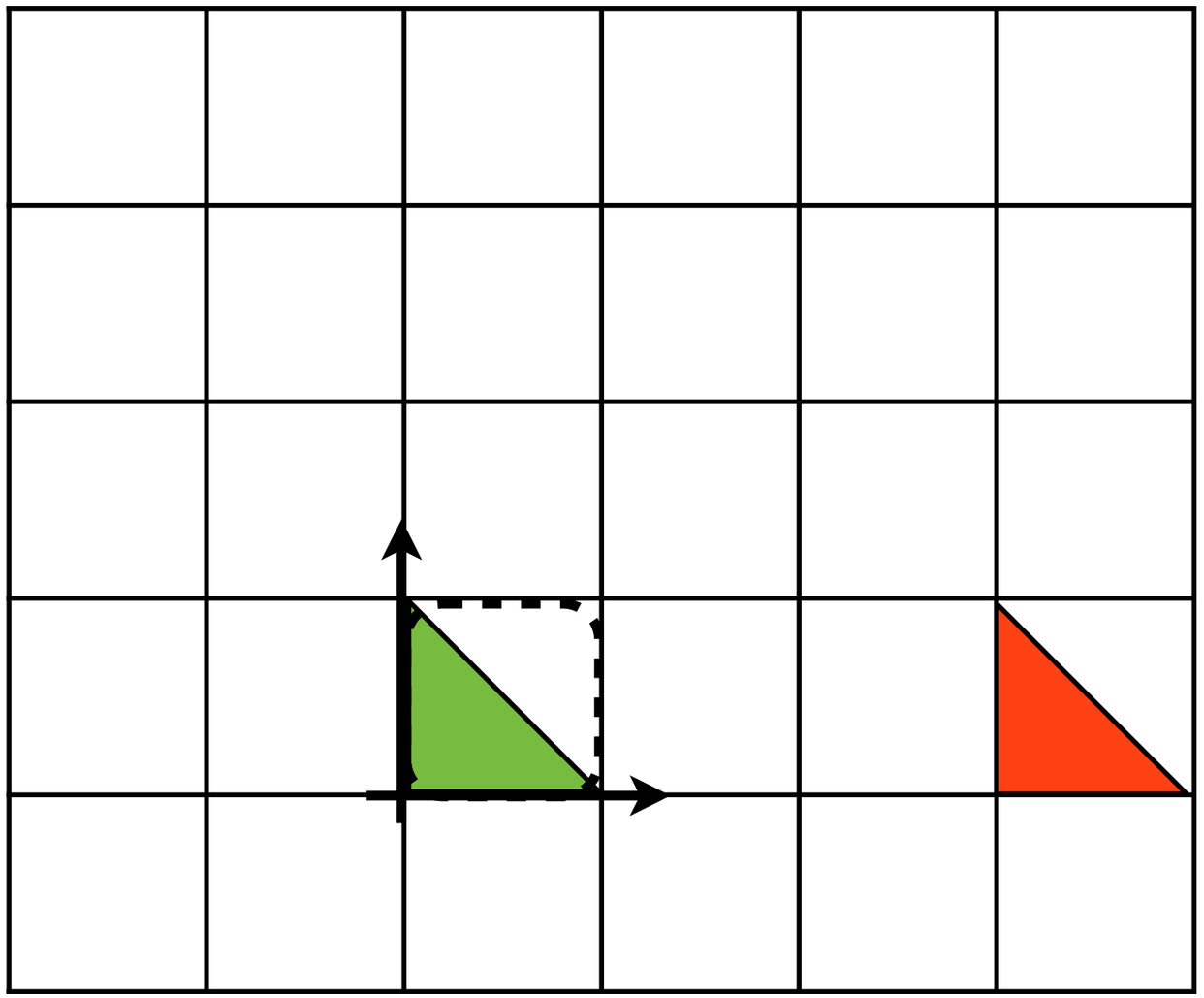
Simple Scene Graph

```
w  = ... // World node
n1 = ... // Red   tri node (0,0,0),(1,0,0),(0,1,0) in model space
n2 = ... // Green tri node (0,0,0),(1,0,0),(0,1,0) in model space
g1 = ... // Group node
w.add_child(n1);
w.add_child(g1);
g1.add_child(n2);
n1.set_translation(3, 0, 0);
n2.set_translation(-1, -1, 0);
g1.set_scaling(2, 1, 1);
g1.set_rotation_z(fPI);
g1.set_translation(1, 1, 0);
```



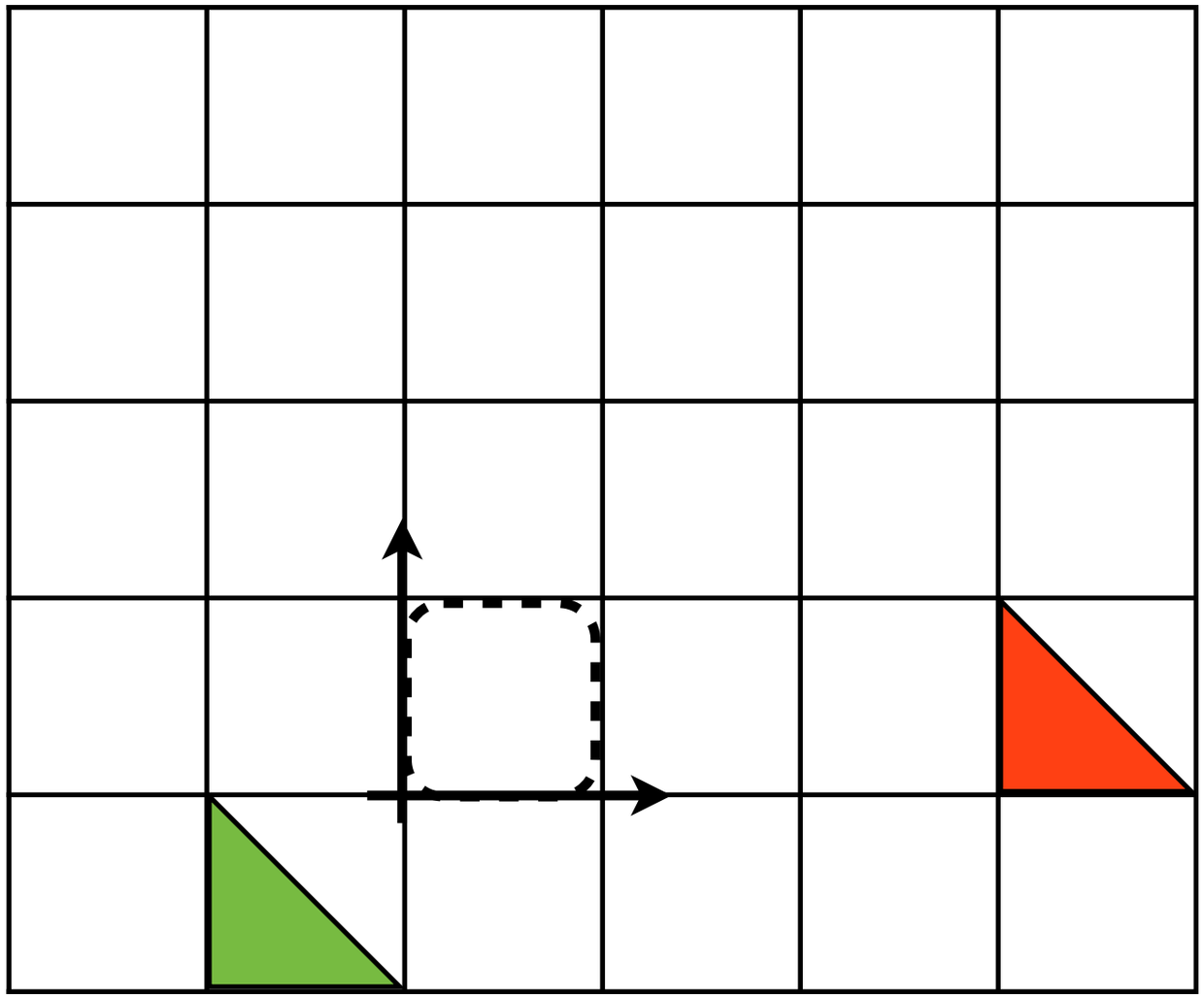
Simple Scene Graph

```
w  = ... // World node
n1 = ... // Red   tri node (0,0,0),(1,0,0),(0,1,0) in model space
n2 = ... // Green tri node (0,0,0),(1,0,0),(0,1,0) in model space
g1 = ... // Group node
w.add_child(n1);
w.add_child(g1);
g1.add_child(n2);
n1.set_translation(3, 0, 0);
n2.set_translation(-1, -1, 0);
g1.set_scaling(2, 1, 1);
g1.set_rotation_z(fPI);
g1.set_translation(1, 1, 0);
```



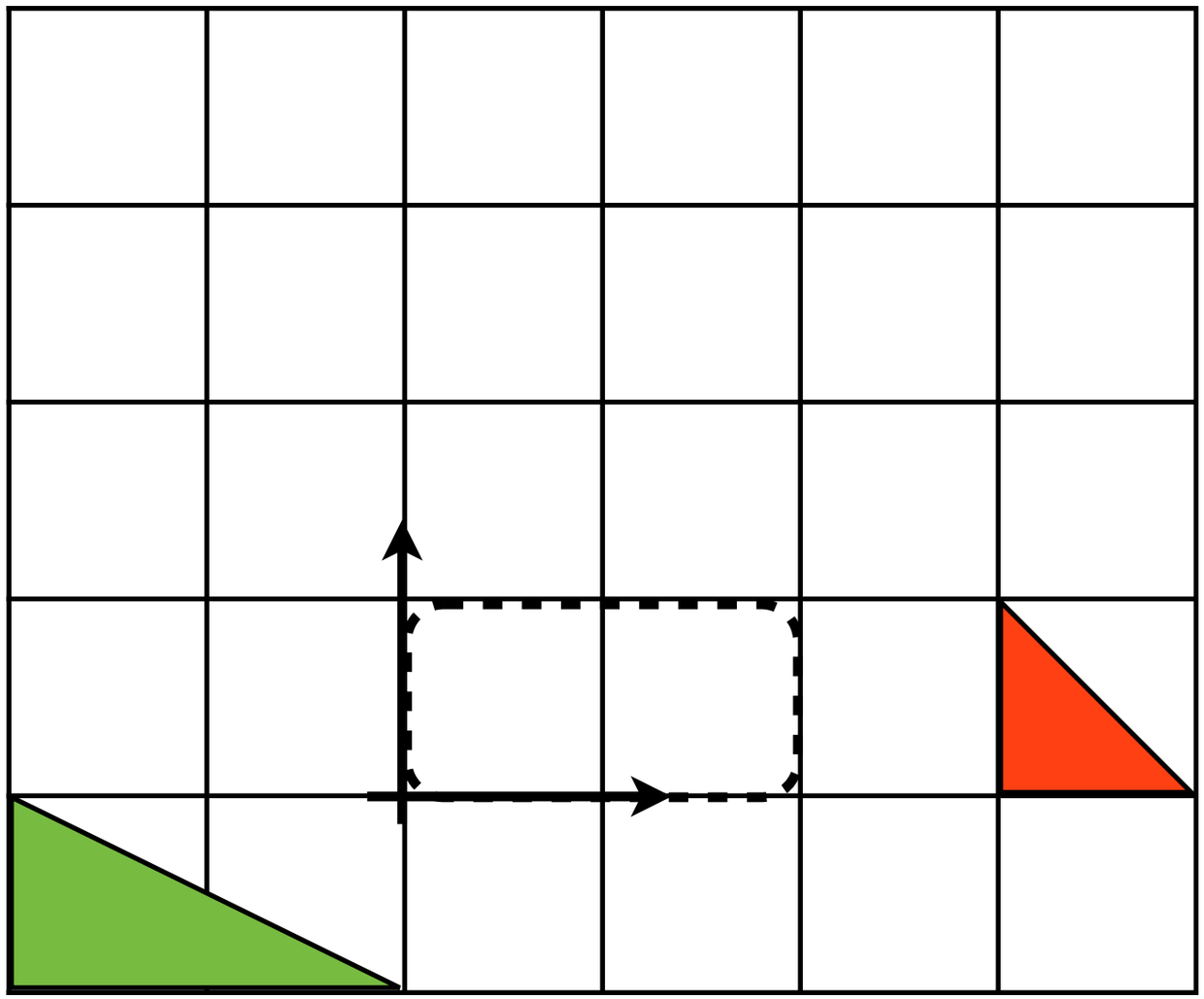
Simple Scene Graph

```
w  = ... // World node
n1 = ... // Red   tri node (0,0,0),(1,0,0),(0,1,0) in model space
n2 = ... // Green tri node (0,0,0),(1,0,0),(0,1,0) in model space
g1 = ... // Group node
w.add_child(n1);
w.add_child(g1);
g1.add_child(n2);
n1.set_translation(3, 0, 0);
n2.set_translation(-1, -1, 0);
g1.set_scaling(2, 1, 1);
g1.set_rotation_z(fPI);
g1.set_translation(1, 1, 0);
```



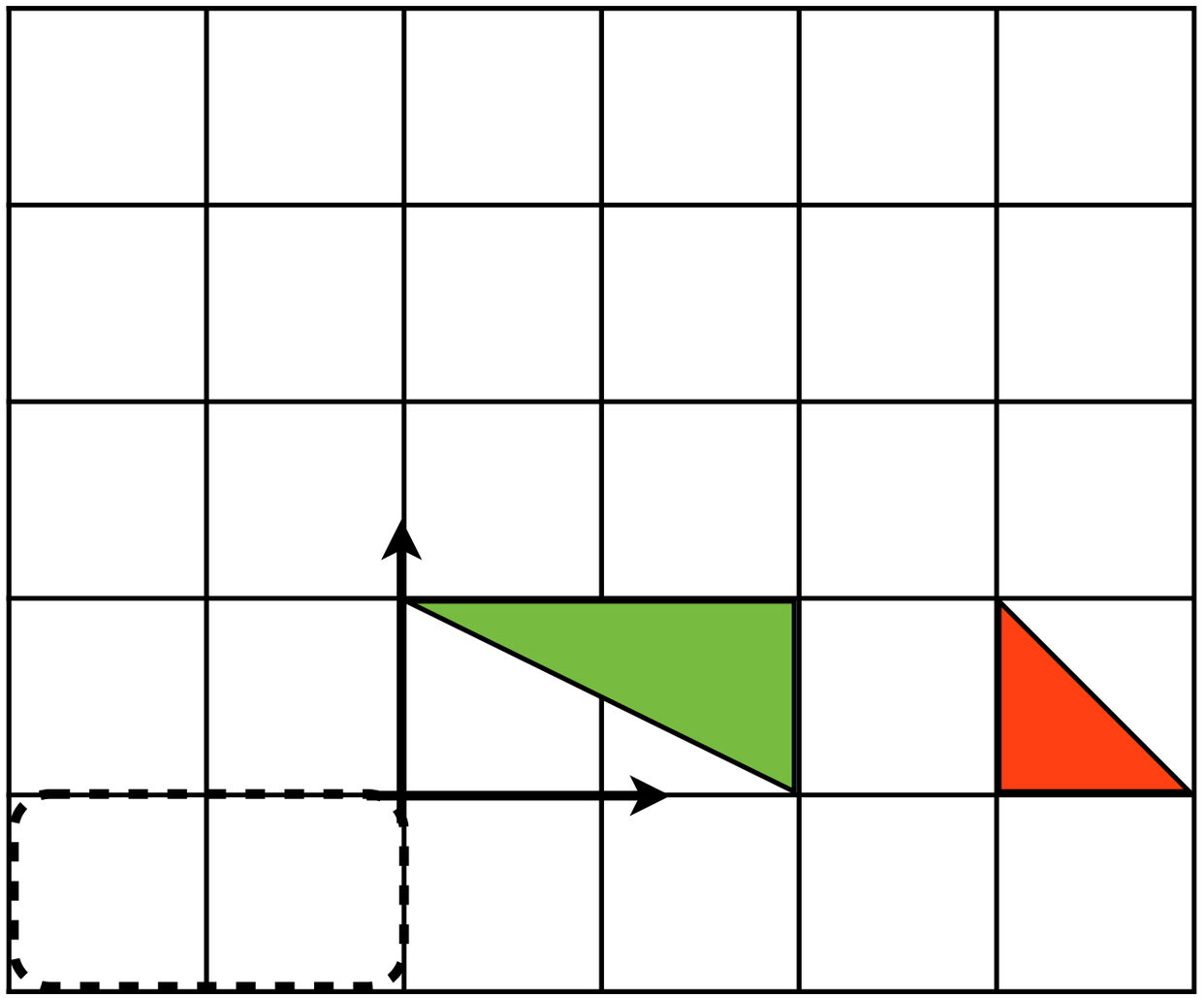
Simple Scene Graph

```
w  = ... // World node
n1 = ... // Red   tri node (0,0,0),(1,0,0),(0,1,0) in model space
n2 = ... // Green tri node (0,0,0),(1,0,0),(0,1,0) in model space
g1 = ... // Group node
w.add_child(n1);
w.add_child(g1);
g1.add_child(n2);
n1.set_translation(3, 0, 0);
n2.set_translation(-1, -1, 0);
g1.set_scaling(2, 1, 1);
g1.set_rotation_z(fPI);
g1.set_translation(1, 1, 0);
```



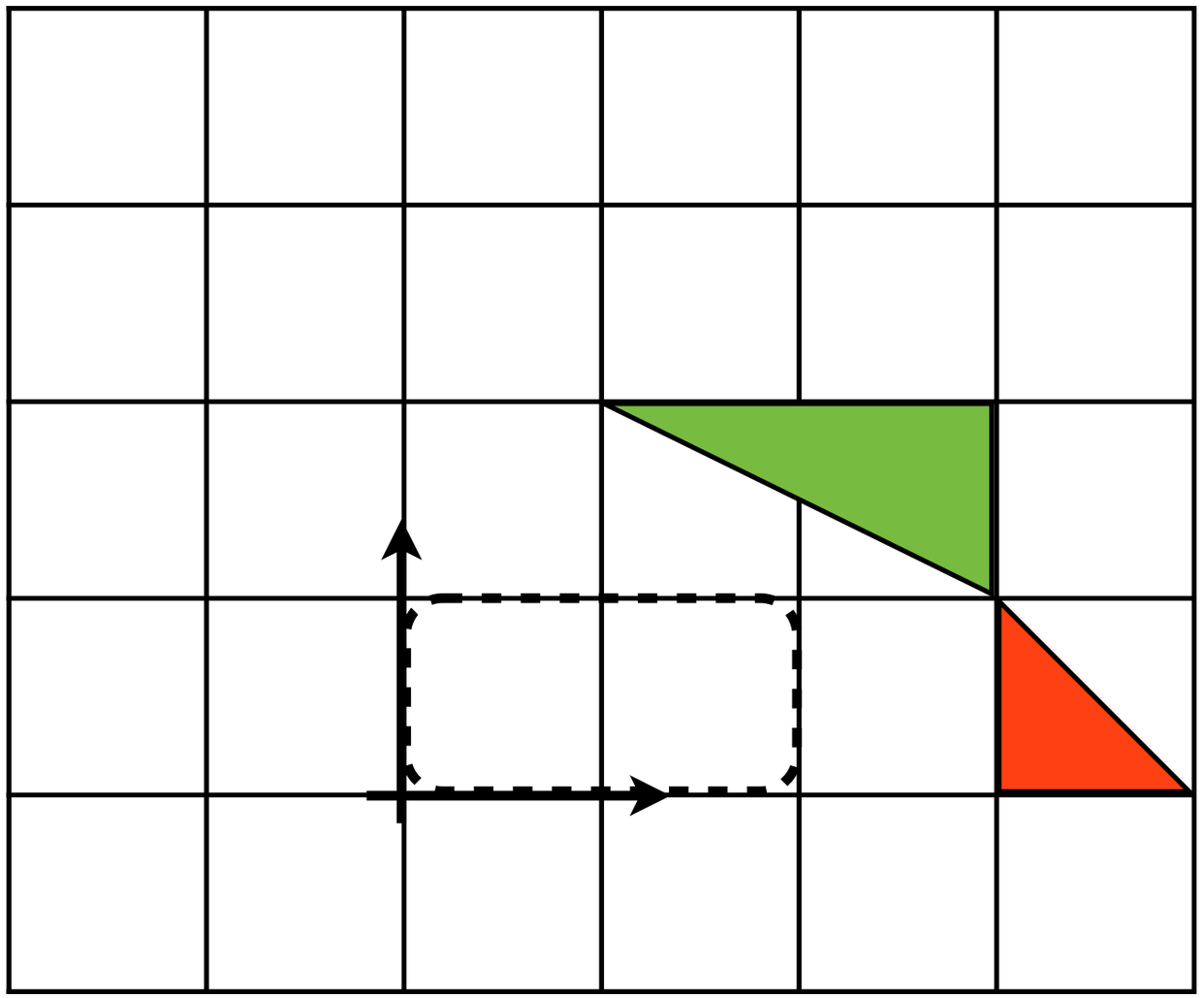
Simple Scene Graph

```
w  = ... // World node
n1 = ... // Red   tri node (0,0,0),(1,0,0),(0,1,0) in model space
n2 = ... // Green tri node (0,0,0),(1,0,0),(0,1,0) in model space
g1 = ... // Group node
w.add_child(n1);
w.add_child(g1);
g1.add_child(n2);
n1.set_translation(3, 0, 0);
n2.set_translation(-1, -1, 0);
g1.set_scaling(2, 1, 1);
g1.set_rotation_z(fPI);
g1.set_translation(1, 1, 0);
```



Simple Scene Graph

```
w  = ... // World node
n1 = ... // Red   tri node (0,0,0),(1,0,0),(0,1,0) in model space
n2 = ... // Green tri node (0,0,0),(1,0,0),(0,1,0) in model space
g1 = ... // Group node
w.add_child(n1);
w.add_child(g1);
g1.add_child(n2);
n1.set_translation(3, 0, 0);
n2.set_translation(-1, -1, 0);
g1.set_scaling(2, 1, 1);
g1.set_rotation_z(fPI);
g1.set_translation(1, 1, 0);
```



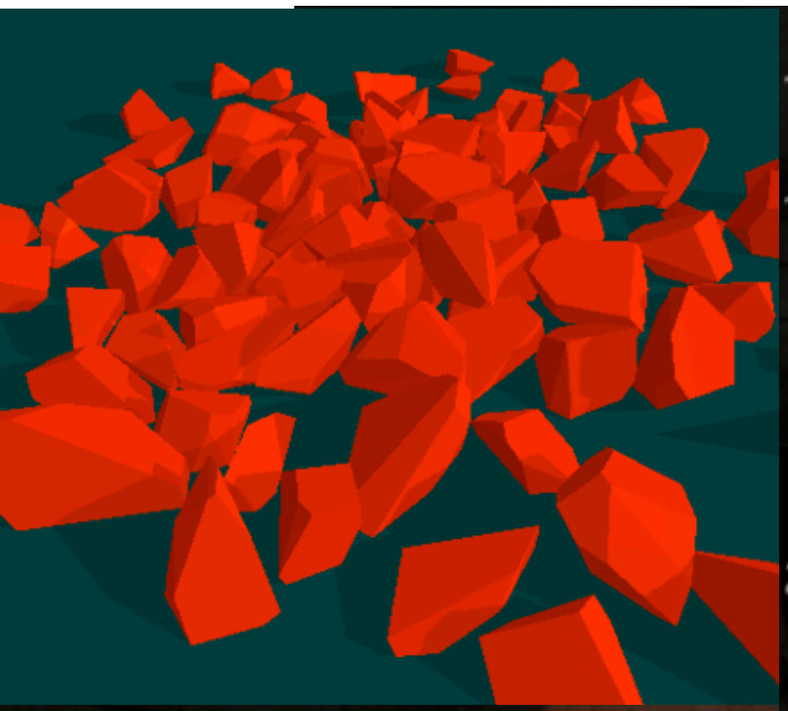
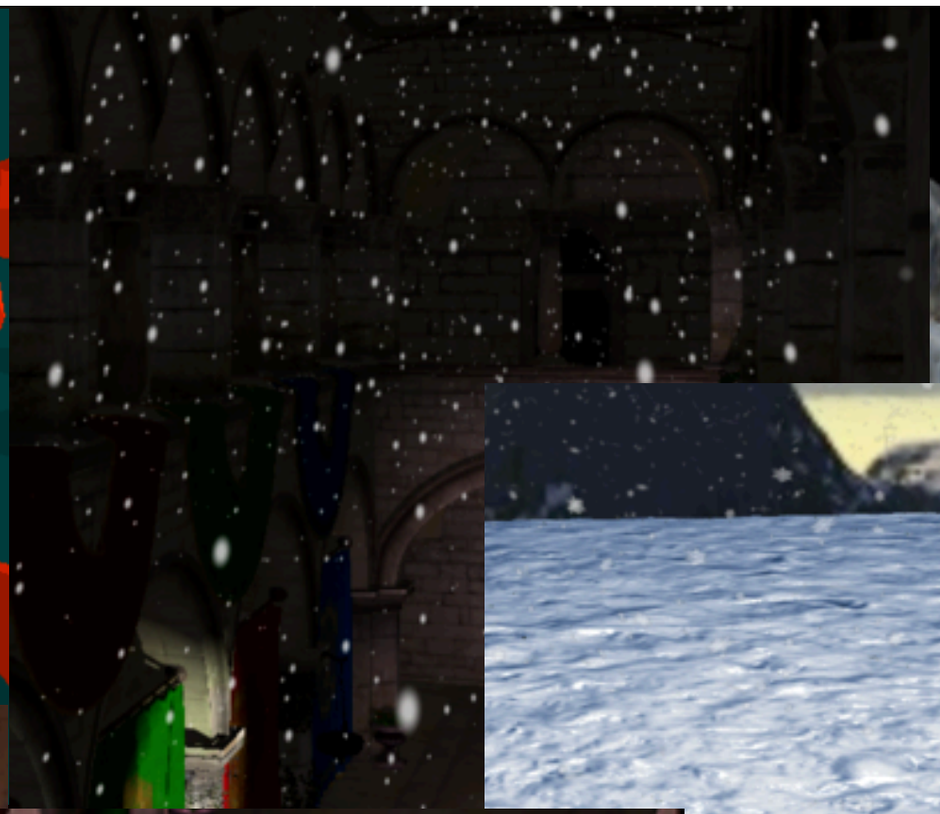
Exercises

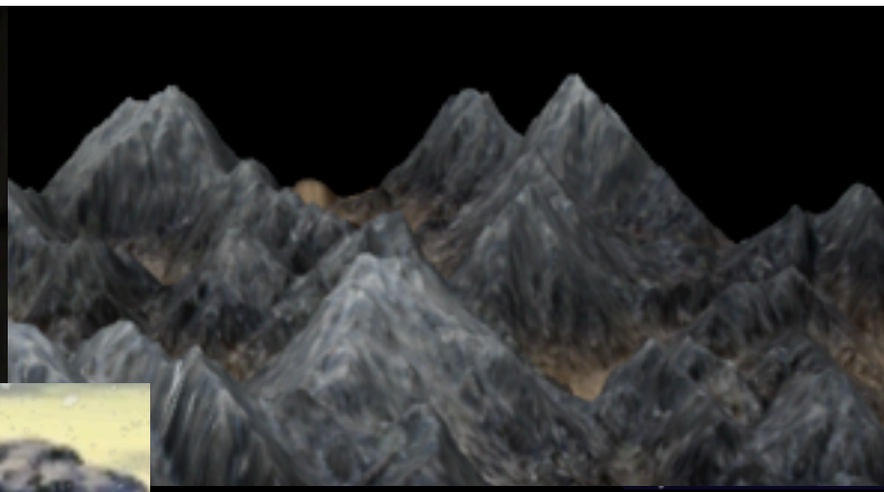
- Some exercises available at the course website
  - https://fileadmin.cs.lth.se/cs/Education/EDA221/lectures/2013/exercise.pdf

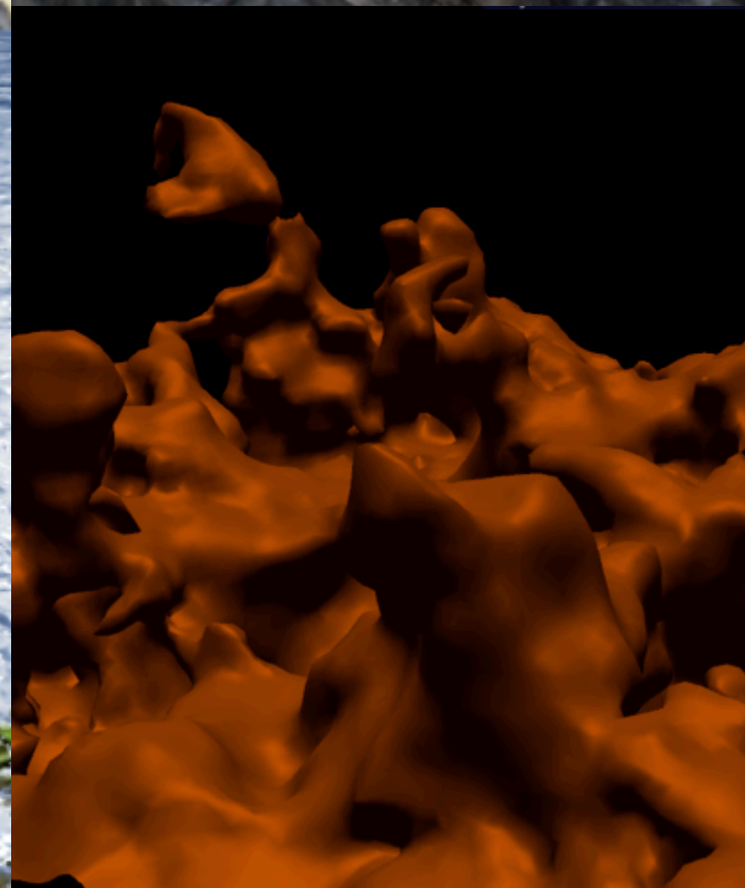
EDAN35 High Performance Computer Graphics

- Fall (HT2)
- Contents:
  - Graphics hardware algorithms
  - GPU programming
    - Using OpenGL
- Two programming assignments
- **Project** with non-compulsory competition
- 7.5 points of fun!

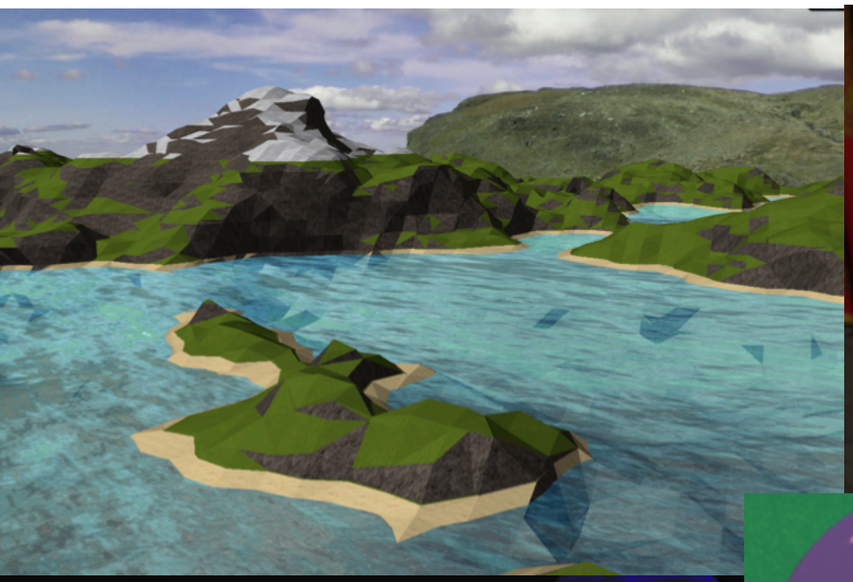
2017 Projects

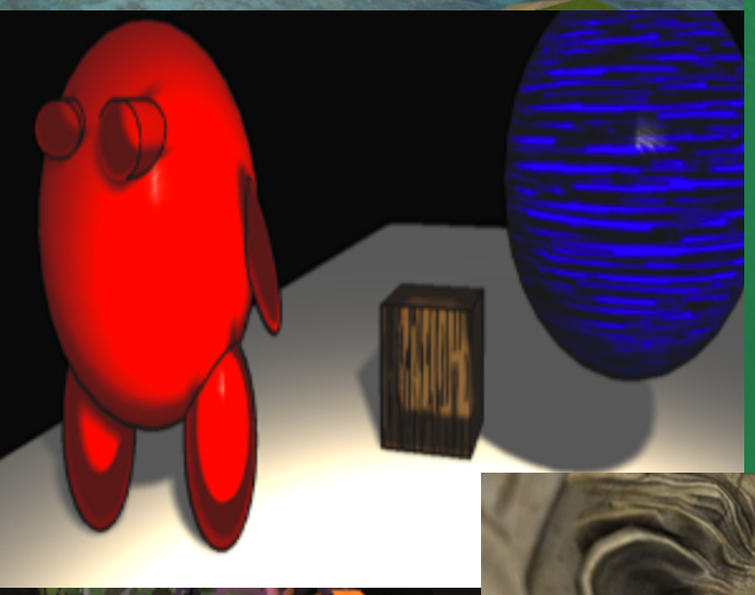



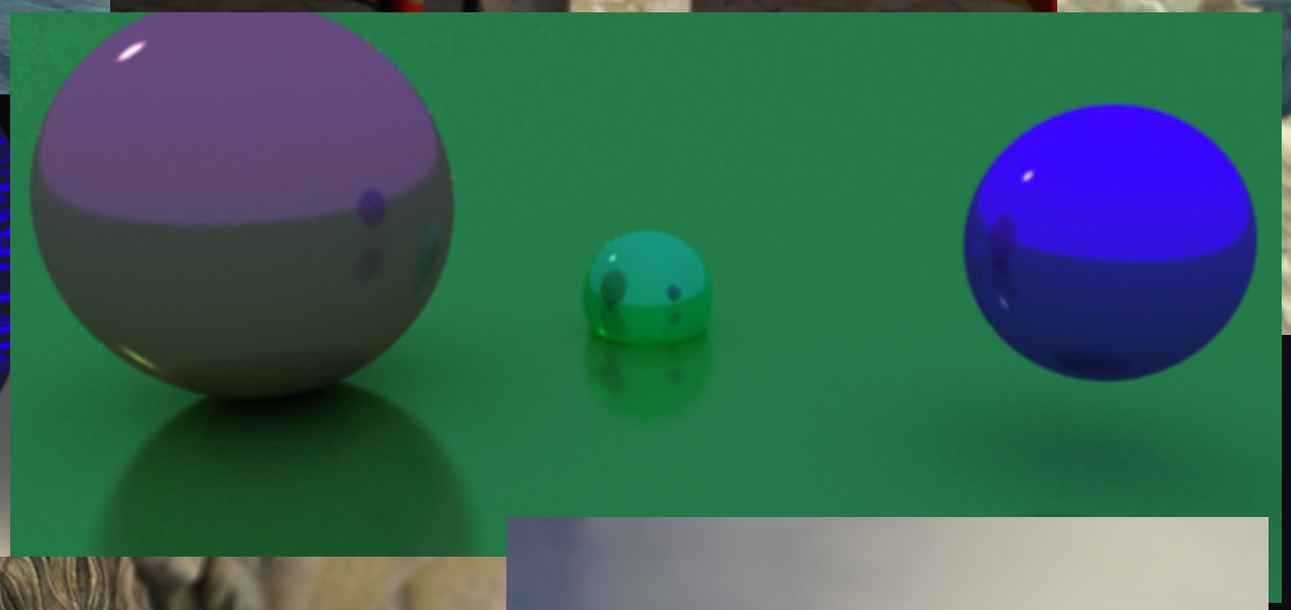


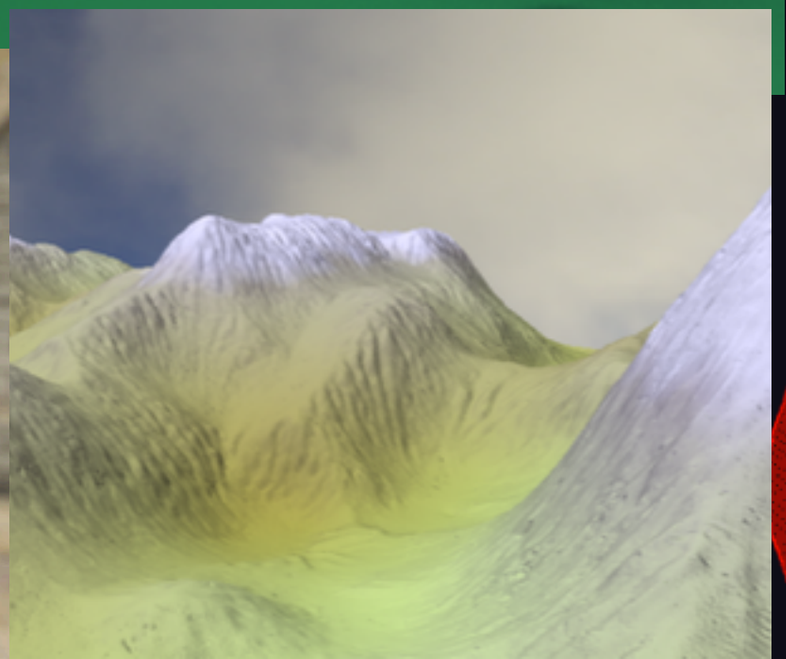


2020 Projects









EDAN35 Project Videos

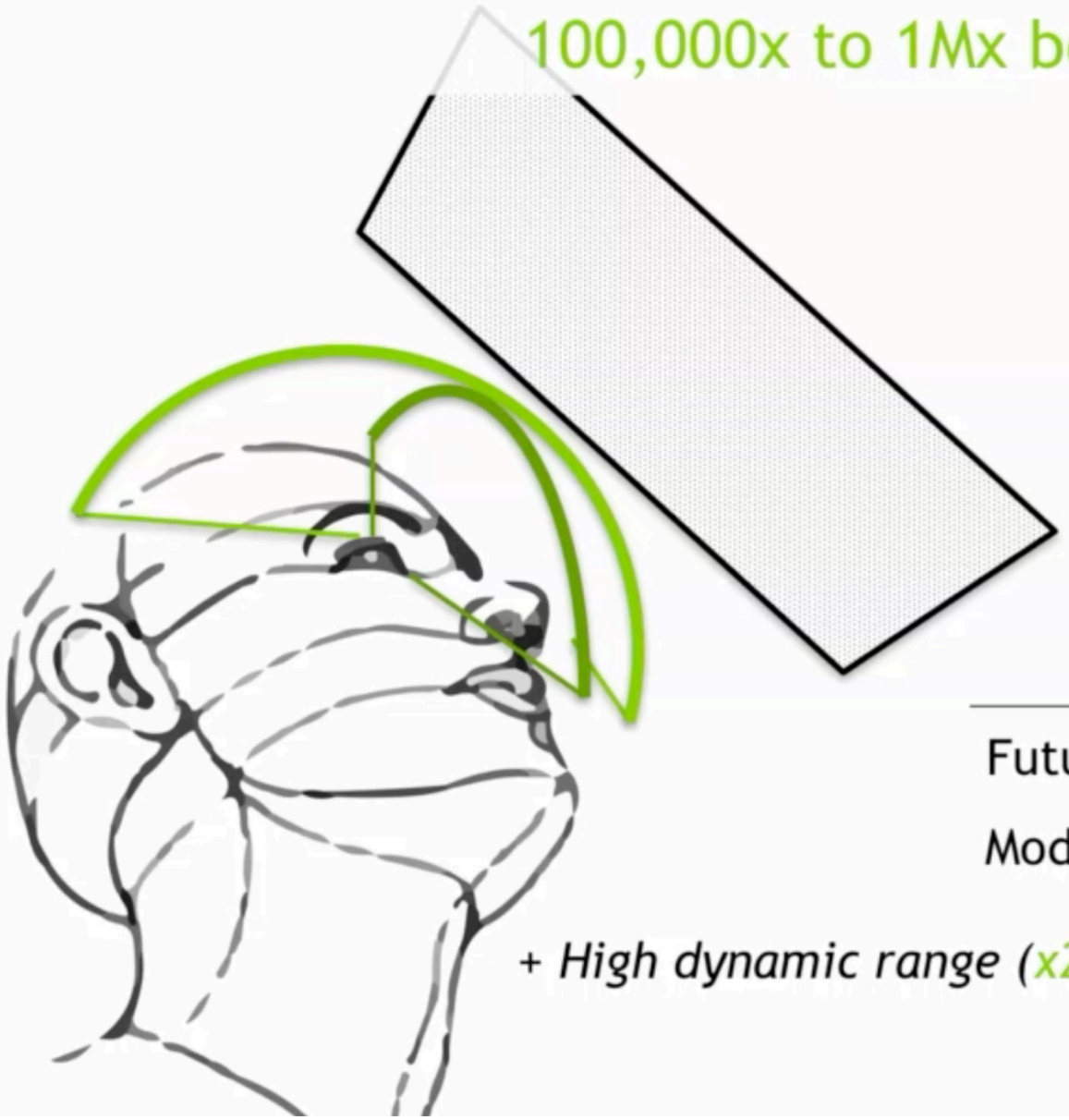
- Frosty and the Flamethrowers, John David Olovsson and Einar Holst, 2010
  - https://youtu.be/UUF_jo33s9Y
- Polyspasm, Jesper Öqvist, 2012
  - https://youtu.be/pSyM80vj-f8
- Lighthouse, Anton Klarén and Valdemar Roxling, 2015
  - https://youtu.be/tFthE2YciRA

EDAN70 Project

- EDAN70 Project in Computer Science
  - LP4 Computer Graphics

LIMITS OF HUMAN PERCEPTION

100,000x to 1Mx beyond modern VR



220° Horizontal
x 135° Vertical

x (120 pixels/degree)^2

≈ 400,000,000 pixels
= 200 x 1080p TVs

x 240 Hz

Future VR = 100,000 Mpix/s

Modern VR = **450** Mpix/s

+ High dynamic range (x2), photorealistic dynamic lighting (x10,000), …

Head image from http://jeffsearle.blogspot.com/2015/09/drawing-head-from-different-angles.html

Slide courtesy Dr. Morgan McGuire, NVIDIA

Research in Computer Graphics

SIGGRAPH 2022 Technical Papers Preview https://youtu.be/sK2EZE617pY

The End!