Midterm Review

Computer Graphics 15-362 / 15-662

Lecture 2: Linear Algebra and Vector Calculus:

- How can we measure vectors?
- What is a Vector Space?
- Draw a geometric representation of each rule that vectors seem to obey.
- Can a function be a vector? Explain
- Add and scale vectors.
- Add and scale functions
- What is the norm of a vector?
- Associate each property of a norm with a geometric interpretation.
- Compute the Euclidean norm in Cartesian coordinates.
- Compute the L2 norm of a function.
- Associate each property of the inner product with a geometric interpretation.
- Compute the inner product in Cartesian coordinates.

Lecture 2: Linear Algebra and Vector Calculus:

- Use the inner product for operations such as projection.
- Compute the inner product of functions.
- Give properties and an example of a linear map.
- Define span and basis..
- Compute an orthonormal basis from a set of vectors.
- Be able to use Gram-Schmidt orthonormalization.
- Know that orthonormalization of functions can be done by decomposing them into sinusoids.
- Be able to solve a simple system of linear equations, depict it geometrically, and represent it in matrix form.
- Be able to represent a linear map in matrix form.

Lecture 2: Linear Algebra and Vector Calculus:

- Euclidean norm is a notion of length preserved by rotations/translations/ reflections of space. Be able to calculate it for a vector of any dimension.
- Compute the inner product of two n-dimensional vectors. What is the geometric meaning if an orthonormal basis is used?
- Compute the cross product of two three-dimensional vectors. What is the geometric meaning of this cross product? What is the geometric meaning of its magnitude?
- Use the cross product to do a quarter rotation of a vector within a plane.
- Represent the dot product using matrix notation.
- Represent the cross product using matrix notation.
- Understand what the determinant measures. What does the determinant of a linear map tell us? Give an example.
- What is a directional derivative?
- Bonus: Compute the gradient of a function.

Lecture 2: Linear Algebra and Vector Calculus:

- Understand gradient as the best linear approximation and the direction of steepest ascent.
- When is the gradient not defined?
- Bonus: Be able to express gradients of simple matrix expressions
- What is a vector field? Give an example.
- Be able to compute divergence, curl, and the Laplacian of a vector field. Also be able to express the meaning of these terms geometrically, for example by drawing a diagram.
- What is the Hessian? Be able to compute the Hessian of a function.

Lecture 4: Transforms

1. Which of the following operations are linear transforms: scale, rotation, shear, translation, reflection, rotation about a point that is not the origin?
2. Express scale as a linear transform
3. Express rotation as a linear transform
4. Express shear as a linear transform
5. Express reflection as a linear transform
6. Express translation as an affine transform
7. Know what makes a transform linear vs. affine
8. Know how to build transformation matrices from start and end configurations of your object

Lecture 4: Transforms

- Create 2D and 3D transformation matrices to perform specific scale, shear, rotation, reflection, and translation operations
- Compose transformations to achieve compound effects
- Rotate an object about a fixed point
- Rotate an object about a given axis
- Create an orthonormal basis given a single vector
- Understand the equivalence of [x y 1] and [wx wy w] vectors
- Explain/illustrate how translations in 2D (x, y) are a shear operation in the homogeneous coordinate space (x, y, w)

Lecture 4 (again): Perspective Projection

- Review:
  - Form an orthonormal basis
  - Create a rotation matrix to rotate any coordinate frame to xyz
  - Create the rotation matrix to rotate the xyz coordinate frame to any other frame
  - Know basic facts about rotation matrices / how to recognize a rotation matrix
    - Rows (also columns) are unit vectors
    - Rows (also columns) are orthogonal to one another
    - If our rows (or columns) are u, v, and w, then uXv=w
    - The inverse of a rotation matrix is its transpose
- For any given setup where we place a camera in the environment, pointing down any of the main coordinate axes (x, y, or z), compute a projection of points in the world onto an image plane.
- Create a projection matrix that projects all points onto an image plane at z=1
- Propose a projection matrix that maintains some depth information

Lecture 4 (again): Perspective Projection

- Understand the motivation behind the projection matrix that projects the view frustum to a unit cube
- Be able to draw / discuss the details of the view frustum
- Prove that a standard projection matrix preserves some information about depth
  Write an algorithm for drawing lines that handles all edge cases (i.e., including edges that are exactly horizontal or vertical).

1. How do we determine coverage for a triangle?
2. Give 2 algorithms for determining triangle coverage
3. Bonus: Be able to use the implicit edge representation to determine if a point is inside a triangle.

Lecture 6: Rasterization

- Textures are used for many things, beyond pasting images onto object surfaces. Be able to list and describe the use cases in this list:
  - Normal maps (create appearance of bumpy object on smooth surface by giving false normal to the lighting equations)
  - Displacement maps (encode offsets in the geometry of a surface, which is difficult to handle in a standard graphics pipeline)
  - (Bonus) Environment maps (store light information in all directions in a scene)
  - (Bonus) Ambient occlusion map (store exposure of geometry to ambient light for better representation of surface appearance with simple lighting models)
  - Can you think of / discover others?
- Know how to interpolate texture coordinates
- Know how to index into a texture and compute a correct color using bilinear interpolation
- Be able to create a mipmap and store it in memory
- Be able to compute color from multiple levels of mipmaps using trilinear interpolation
- What is the logic behind selecting an appropriate level in a mipmap?
- What can happen if we select a level that is too high resolution? too low resolution?

Lecture 6: Rasterization

1. How should we choose the correct color for a pixel? There is not an exact right answer. However, you should be able to discuss some of the issues involved.
2. What is aliasing, and what artifacts does it produce in our images and our animations? Give at least 3 different examples.
3. One form of aliasing is where high frequencies masquerade as low frequencies. Give an example of this phenomenon.
4. Suppose we have a single red triangle displayed against a blue background. Does this scene contain high frequencies?
5. (Bonus) What does the Nyqvist-Shannon theorem tell us about how image frequencies relate to required sampling rate?
6. (Bonus) One practical solution on your graphics card for reducing aliasing (i.e., for antialiasing) is to take multiple samples per pixel and average to get pixel color. Try to use sampling theory to explain as precisely as you can why taking multiple samples per pixel can reduce aliasing artifacts

Lecture 7: Depth and Blending

- Interpolate colors using barycentric coordinates
- Bonus: Compute barycentric coordinates of a point using implicit edge functions
- Compute barycentric coordinates of a point using triangle areas
- Estimate the location of a point inside a triangle given its barycentric coordinates
- Estimate the location of a point outside a triangle given its barycentric coordinates
- Estimate barycentric coordinates of a point from a drawing.
- Show that interpolation in 3D space followed by projection can give a different result from projection followed by interpolation in screen space. In other words, explain why interpolation using barycentric coordinates in screen space may give a result that is incorrect.
- How, then, can we obtain a correct result using interpolation in screen space?

Lecture 7: Depth and Blending

- What is the depth buffer (Z-buffer) and how is it used for hidden surface removal?
- Where does the depth for each sample / fragment come from? Where is it computed in the graphics pipeline?
- Is the depth represented in the depth buffer the actual distance from the camera? If not, what is it?
- What is the meaning of the alpha parameter in the [R G B a] color representation?
- Be able to use alpha to do compositing with the "Over" operator.
- Is "Over" commutative? If not, create a counterexample.
- What is premultiplied alpha, and how does it work?
- Be able to use premultiplied alpha for "Over" composition.
- Why is premultiplied alpha better? Give 2 examples.
- How do we properly render a scene with mixed opaque and semi-transparent triangles? What is the rendering order we should use? When is the depth buffer updated?
- Draw a rough sketch of the graphics pipeline. Think about transforming triangles into camera space, doing perspective projection, clipping, transforming to screen coordinates, computing colors for samples, computing colors for pixels, the depth test, updating color and depth buffers.

$$C = \alpha_B B + (1 - \alpha_B)\alpha_A A$$

Lecture 7: 3D Rotations (Bonus)

- What is the problem of gimbal lock? Give an example where this problem occurs.
- How does using quaternions solve this problem?
- Know that every rotation can be expressed as rotation by some angle about some axis.
- Know how to go between quaternions and axis-angle format for rotations.
- Know that quaternions are expressed as higher dimensional complex numbers.
- Be able to work out quaternion multiplication from the complex number representation of a quaternion.

Midterm Review

Midterm Overview

- 80 minutes, during class this Wednesday
  - Basic mathematical questions, no intense calculation
  - Know your definitions and be able to apply them!
  - No pseudocode
  - Review slides are a good hint as to what might be on the exam :)

- Cheat sheet: one 3x3 inch note (about the size of a post it note) front and back

- **Please bring a blue/black pen to write your solutions**

- Transformations Review
- Rasterization Review

Transformations

- Homogeneous coordinates
- 3D Translation
- 3D Scale
- 3D Rotation
  - Axis-Aligned rotation
  - Axis-Angle rotation
  - Rotations from orthonormal bases

3D Transforms in Homogeneous Coordinate

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

[point in 3D]

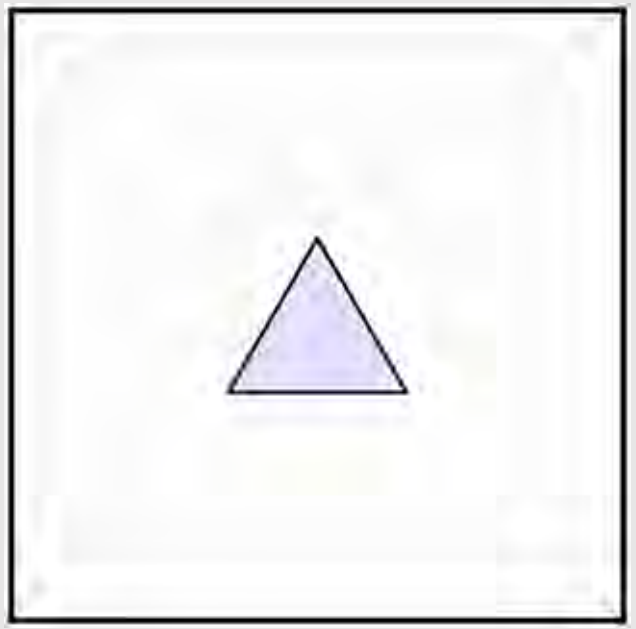
Matrix representations of 3D linear transformations just get an additional identity row/column:

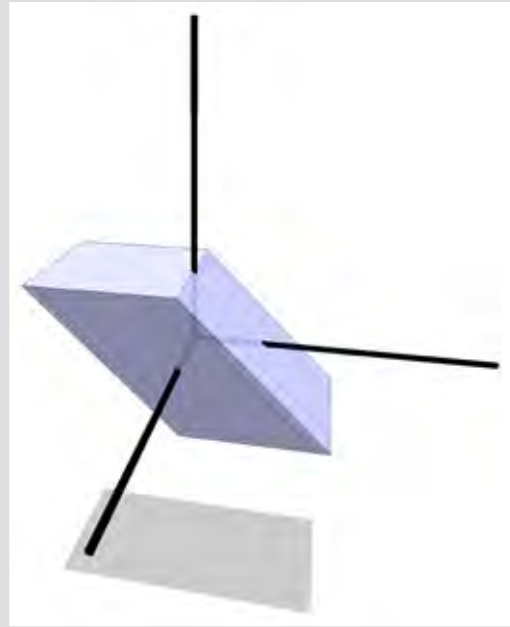
$$\begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & s & 0 \\ 0 & 1 & t & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} a & 0 & 0 & 0 \\ 0 & b & 0 & 0 \\ 0 & 0 & c & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & 0 & u \\ 0 & 1 & 0 & v \\ 0 & 0 & 1 & w \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

[rotate around y by θ] [shear by z in (s,t) direction] [scale by a,b,c] [translate by (u,v,w)]

Translation in Homogeneous Coordinates

- A 2D translation is similar to a 3D shear
  - Moving a slice up/down the shear moves the shape





- Recall shear is written as:

$$f_{\mathbf{u},\mathbf{v}}(\mathbf{x}) = \mathbf{x} + \langle \mathbf{v}, \mathbf{x} \rangle \mathbf{u}$$

$$f_{\mathbf{u},\mathbf{v}}(\mathbf{x}) = \left(I + \mathbf{u}\mathbf{v}^{\mathsf{T}}\right)\mathbf{x}$$

- In our case, $v = (0, 0, 1)$, so**

$$\begin{bmatrix} 1 & 0 & u_1 \\ 0 & 1 & u_2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} cp_1 \\ cp_2 \\ c \end{bmatrix} = \begin{bmatrix} c(p_1 + u_1) \\ c(p_2 + u_2) \\ c \end{bmatrix} \xrightarrow{1/c} \begin{bmatrix} p_1 + u_1 \\ p_2 + u_2 \end{bmatrix}$$

**most often in this class we will also use c = 1

Non-Uniform Scaling

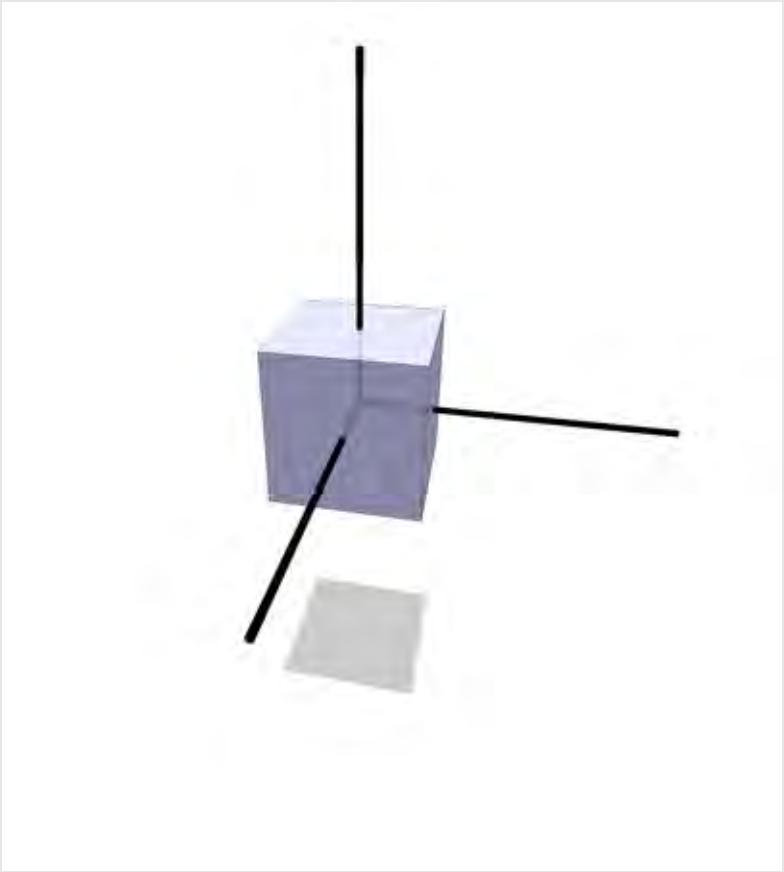
- To scale a vector u by a non-uniform amount (a, b, c):

$$\begin{bmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & c \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} = \begin{bmatrix} au_1 \\ bu_2 \\ cu_3 \end{bmatrix}$$

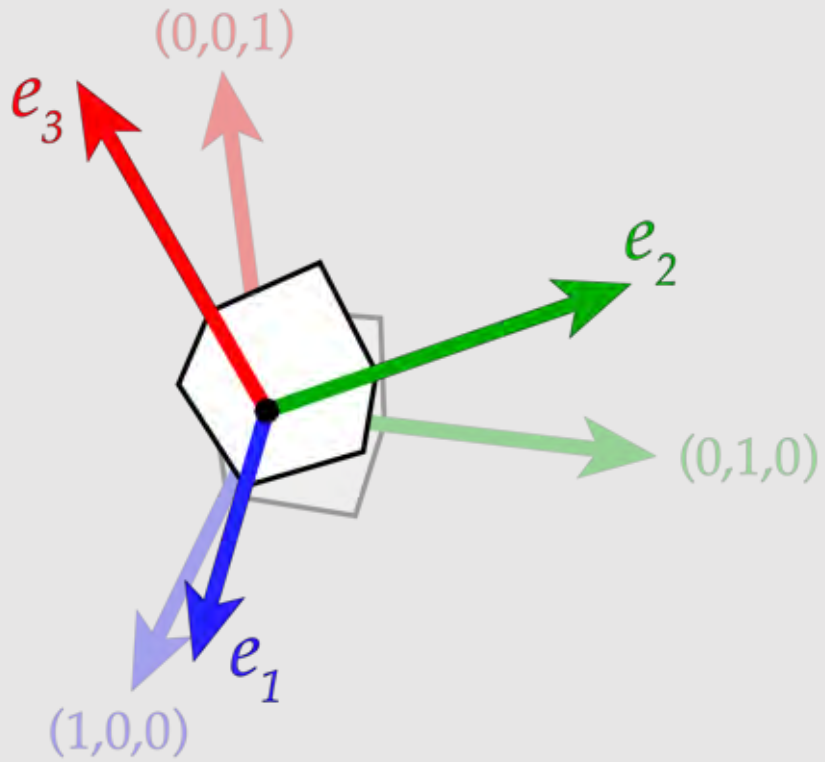
- The above works only if scaling is axis-aligned. What if it isn't?
- Idea:
  - Rotate to a new axis R
  - Perform axis-aligned scaling D
  - Rotate back to original axis R^T

$$A := R^T D R$$

  - Resulting transform A is a symmetric matrix



3D Inverse Rotations



$$R^T \qquad R$$

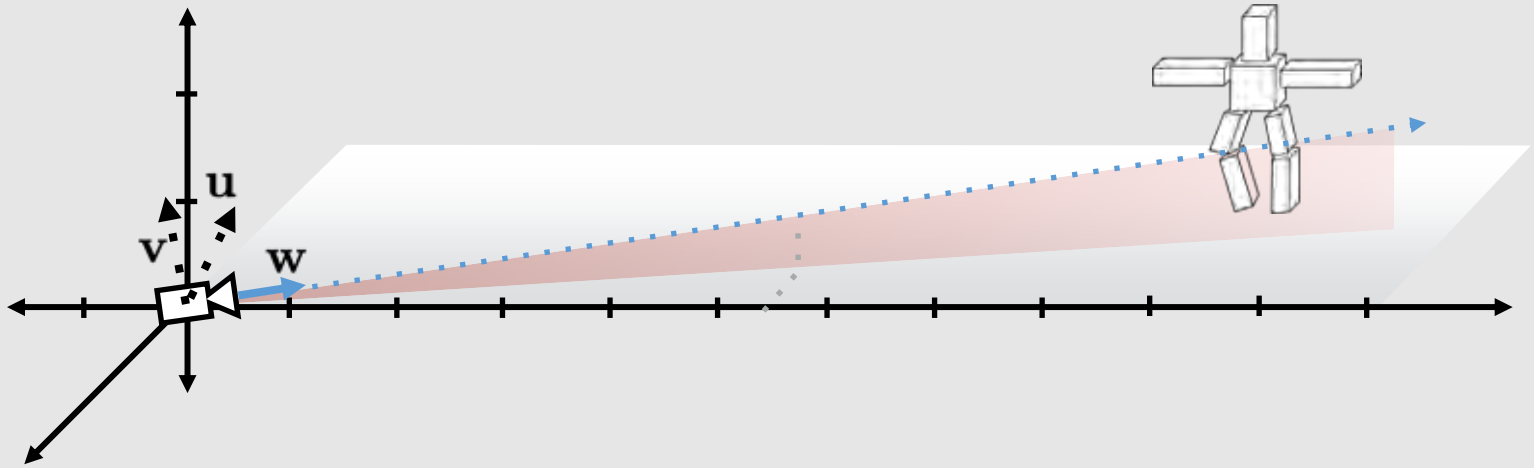
$$\begin{bmatrix} — & e_1^T & — \\ — & e_2^T & — \\ — & e_3^T & — \end{bmatrix} \begin{bmatrix} | & | & | \\ e_1 & e_2 & e_3 \\ | & | & | \end{bmatrix}$$

$$= \begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix} = \begin{bmatrix} e_1^T e_1 & e_1^T e_2 & e_1^T e_3 \\ e_2^T e_1 & e_2^T e_2 & e_2^T e_3 \\ e_3^T e_1 & e_3^T e_2 & e_3^T e_3 \end{bmatrix}$$

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

$$R^T R = I \;\Rightarrow\; R^T = R^{-1}$$

Rotations From Orthonormal Bases

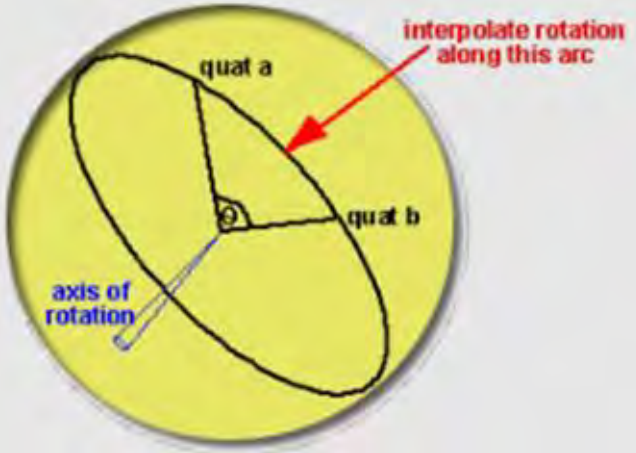


$$R = \begin{bmatrix} -u_x & v_x & -w_x \\ -u_y & v_y & -w_y \\ -u_z & v_z & -w_z \end{bmatrix} \qquad R^{-1} = \begin{bmatrix} -u_x & -u_y & -u_z \\ v_x & v_y & v_z \\ -w_x & -w_y & -w_z \end{bmatrix}$$

Other Ways to Rotate - Euler Angles and Quaternions

- **Euler Angles:** Rotate by combining rotation matrices for each major axis
  - Rx = rotation about x axis, Ry = rotation about y axis, Rz =rotation about z axis
    - RxRyRz = final rotation matrix
  - Fairly intuitive, but prone to gimbal lock!
  - Interpolating between Euler angles can create odd looking path, non-uniform rotation speed, etc.

- **Quaternions**: 4 coordinates, 1 real and 3 complex
  - Defined by $i^2 = j^2 = k^2 = ijk = -1$
  - Not prone to gimbal lock! But not very intuitive to think about :(
  - Can interpolate between them correctly with SLERP



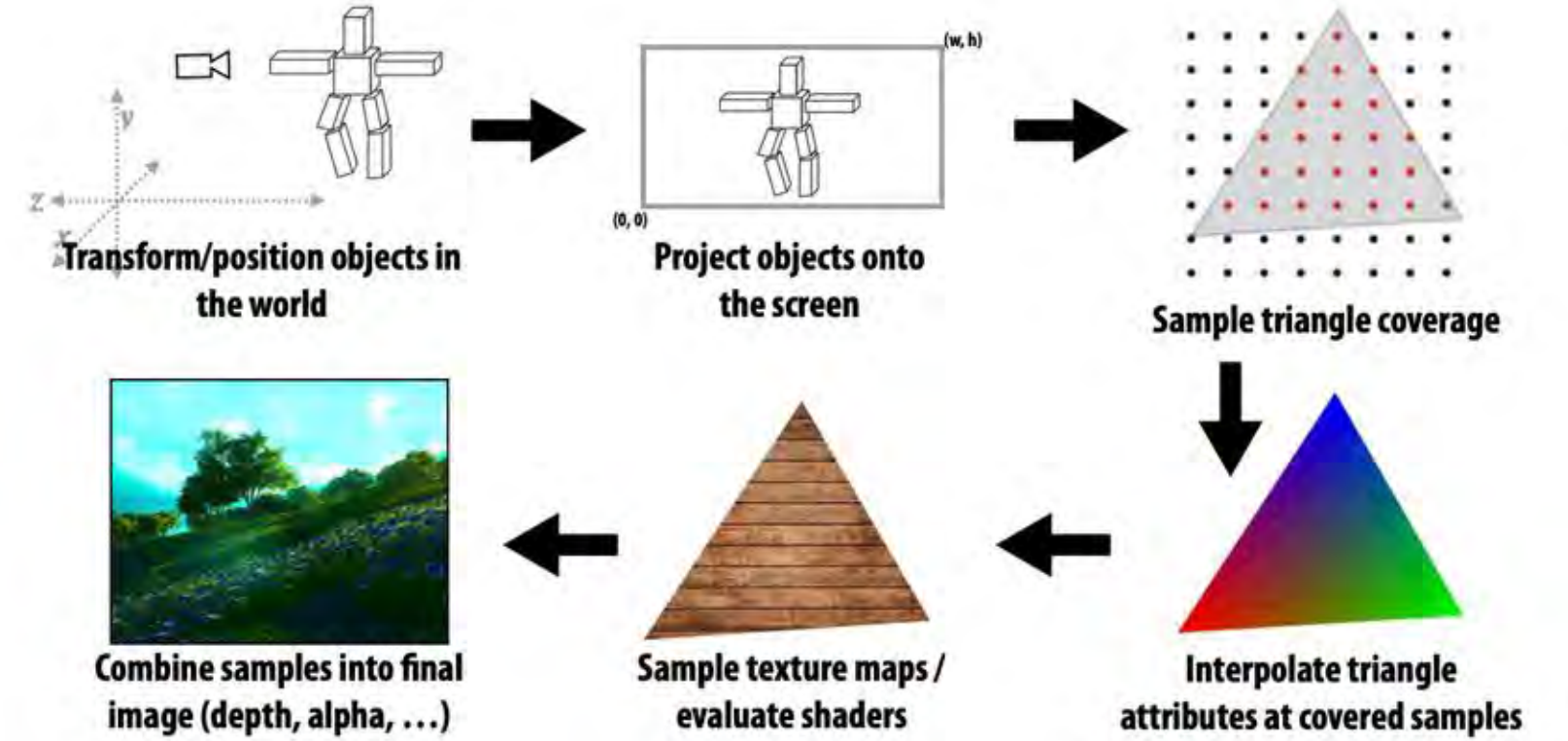
Animating Rotation with Quaternion Curves (1985) Shoemake

- Transformations Review
- Rasterization Review

Rasterization

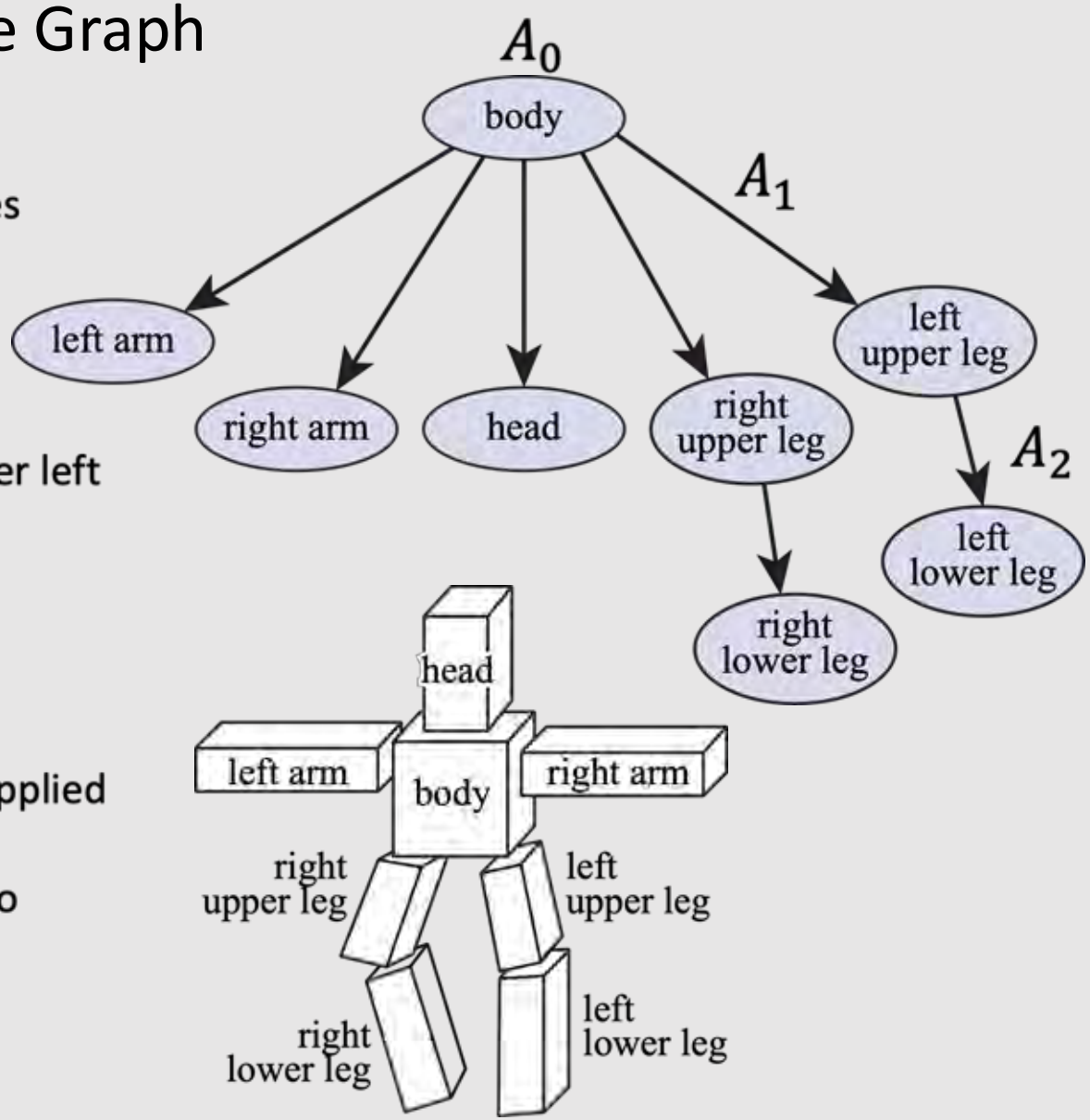
- The “simpler” graphics pipeline
- Scene graph
- Clipping
- Rasterization
  - Sampling
  - Point-in-triangle tests
  - Barycentric coordinates
- Textures
- Depth and Alpha blending

The “Simpler” Graphics Pipeline



Scene Graph

- Suppose we want to build a skeleton out of cubes
- **Idea:** transform cubes in world space
  - Store transform of each cube
- **Problem:** If we rotate the left upper leg, the lower left leg won't track with it
  - **Better Idea:** store a hierarchy of transforms
    - Known as a scene graph
    - Each edge (+root) stores a linear transformation
    - Composition of transformations gets applied to nodes
      - Keep transformations on a stack to reduce redundant multiplication
- **Lower left leg transform:** $A_2 A_1 A_0$

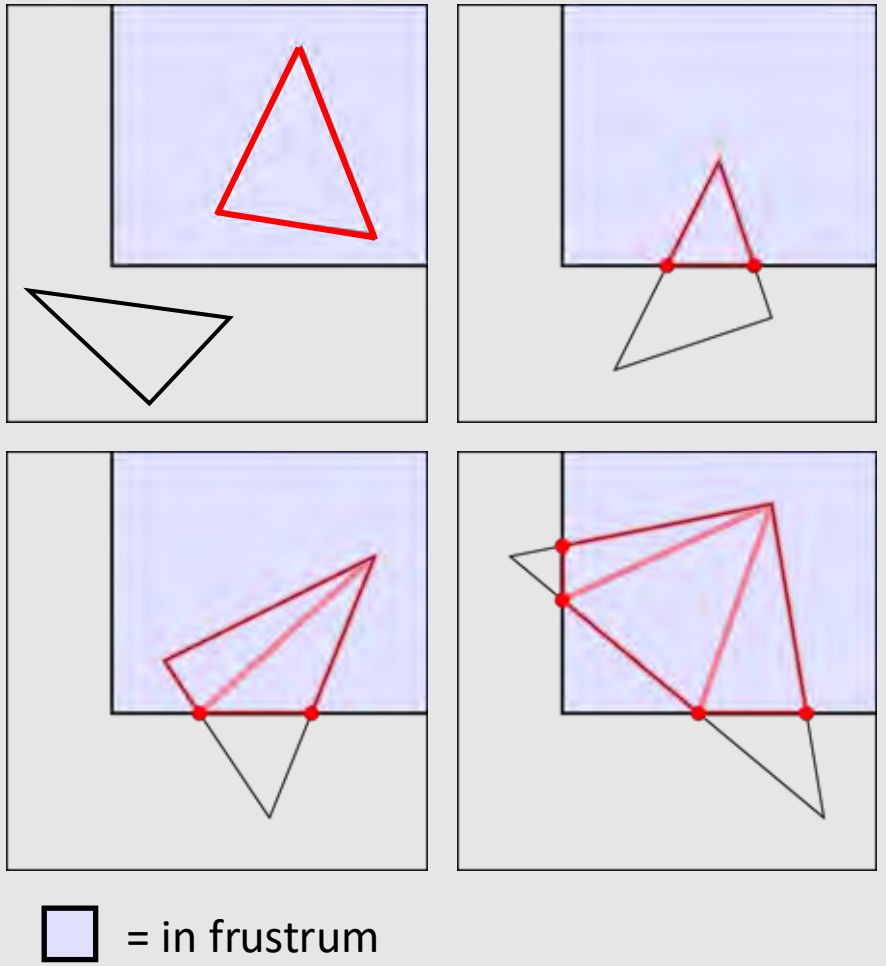


Clipping

- Clipping eliminates triangles not visible to the camera (not in view frustum)
  - Don't waste time rasterizing primitives you can't see!
  - Discarding individual fragments is expensive
    - "Fine granularity"
  - Makes more sense to toss out whole primitives
    - "Coarse granularity"

- What if a primitive is **partially clipped?**
  - Partially enclosed primitives are triangulated into non-overlapping smaller triangles that fit in the frustrum

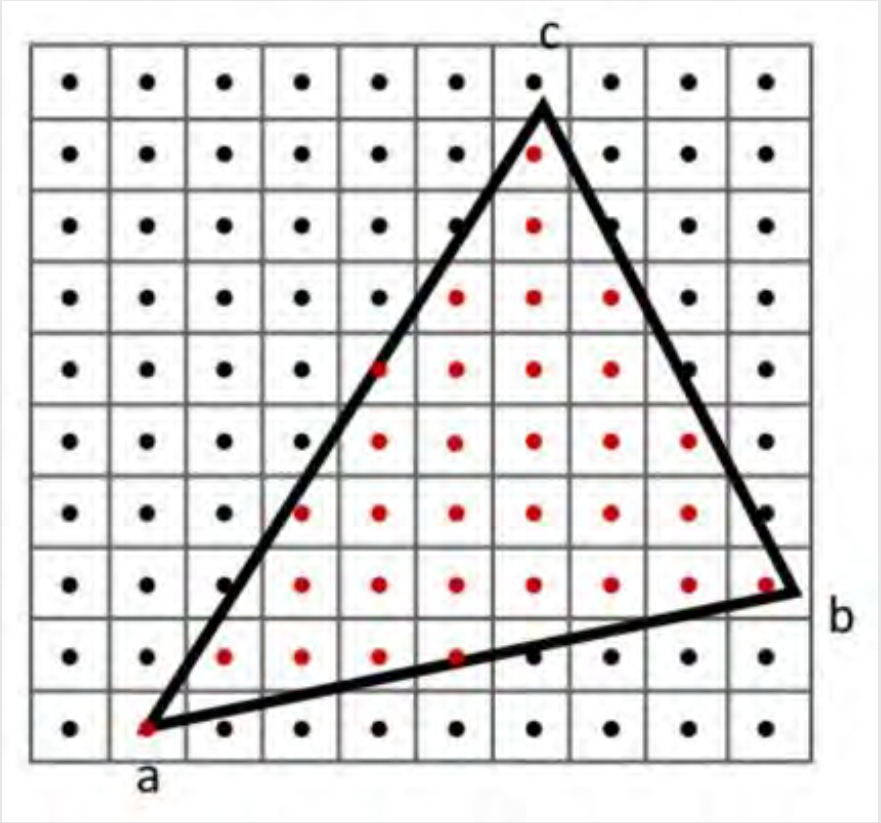
- If part of a triangle is outside the frustrum, it means at least one of its vertices are outside the frustrum
  - **Idea:** check which side of halfspaces the vertices are at



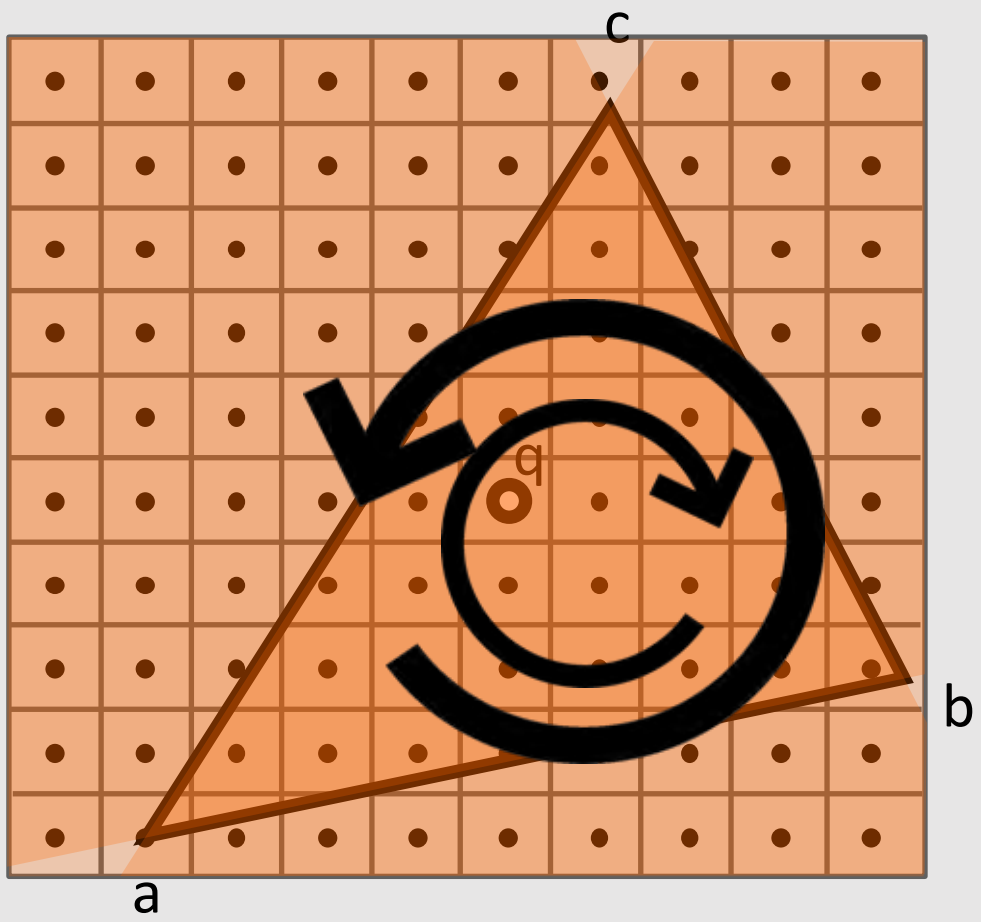
Rasterization

- Triangle
  - Bounding box
  - Incremental triangle traversal
  - Hierarchical coverage

- For each **Primitive** (Triangle):
  - For each **Pixel**:
    - If **Pixel** in **Primitive**:
      - Pixel color = Interpolated triangle color



Point-In-Triangle Test



- **Measurements must all either be positive or negative** for point to be in triangle

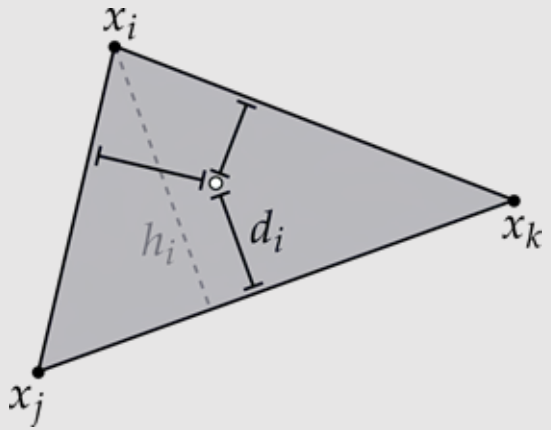
$$(\overrightarrow{ac} \times \overrightarrow{ab}) \cdot (\overrightarrow{ac} \times \overrightarrow{aq}) > 0 \;\&\&$$
$$(\overrightarrow{cb} \times \overrightarrow{ca}) \cdot (\overrightarrow{cb} \times \overrightarrow{cq}) > 0 \;\&\&$$
$$(\overrightarrow{ba} \times \overrightarrow{bc}) \cdot (\overrightarrow{ba} \times \overrightarrow{bq}) > 0$$

OR

$$(\overrightarrow{ab} \times \overrightarrow{ac}) \cdot (\overrightarrow{ac} \times \overrightarrow{aq}) < 0 \;\&\&$$
$$(\overrightarrow{ca} \times \overrightarrow{cb}) \cdot (\overrightarrow{cb} \times \overrightarrow{cq}) < 0 \;\&\&$$
$$(\overrightarrow{bc} \times \overrightarrow{ba}) \cdot (\overrightarrow{ba} \times \overrightarrow{bq}) < 0$$

- Orientation no longer matters
  - Just be consistent!

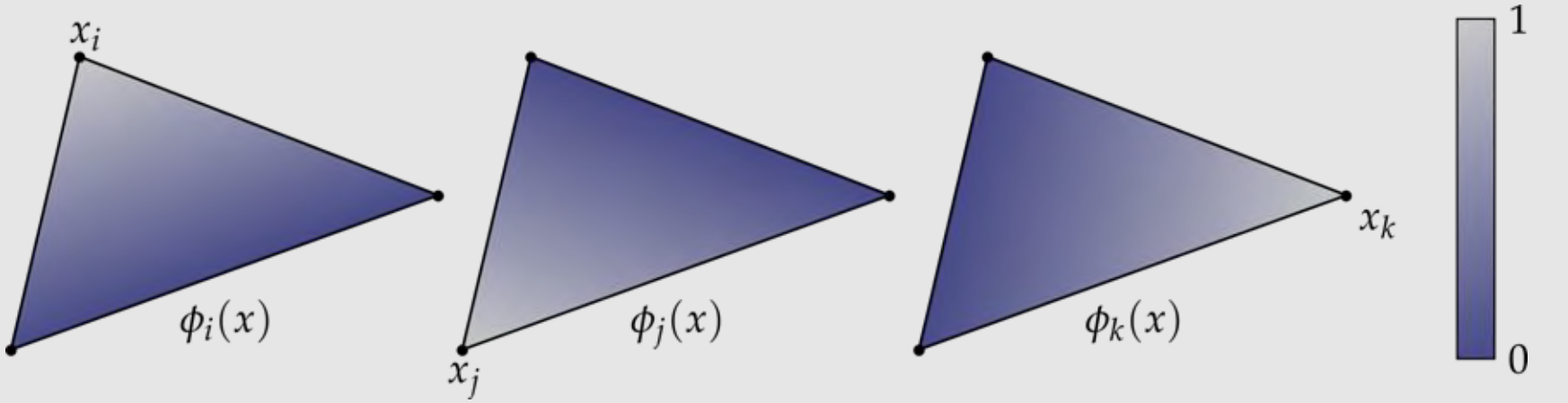
Barycentric Coordinates



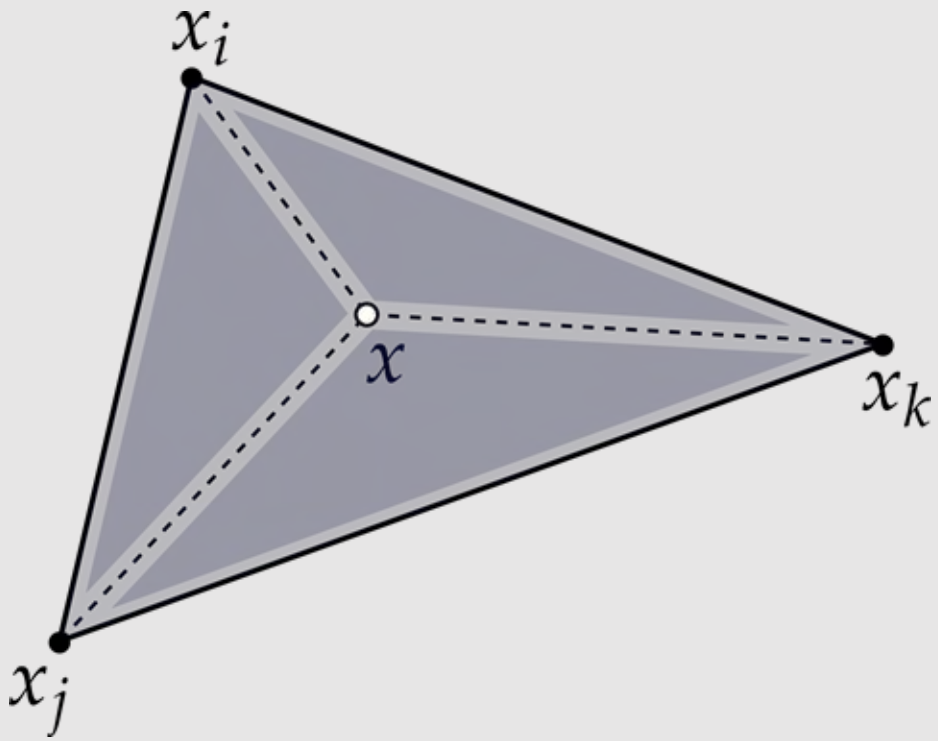
- Inversely proportional to the signed distance between the target point and a point within the triangle
- Can be computed as:

$$\phi_i(x) = d_i(x)/h_i$$

- How would you compute h_i? $d_i(x)$?



Barycentric Coordinates [Another Way]



- Directly proportional to the signed area created by the triangle composed of the other two target points and a point within the triangle
- Can be computed as:

$$\phi_i(x) = \frac{\text{area}(x, x_j, x_k)}{\text{area}(x_i, x_j, x_k)}$$

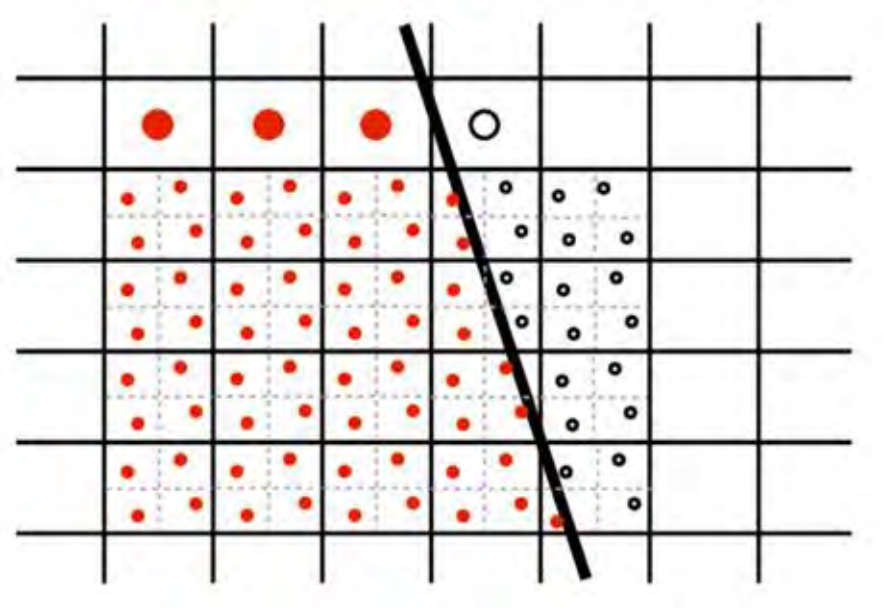
- Note that signed distance / area implies barycentric coordinates can be negative, but they will still sum to 1! (if on the same plane, otherwise we project point to the plane containing our triangle)

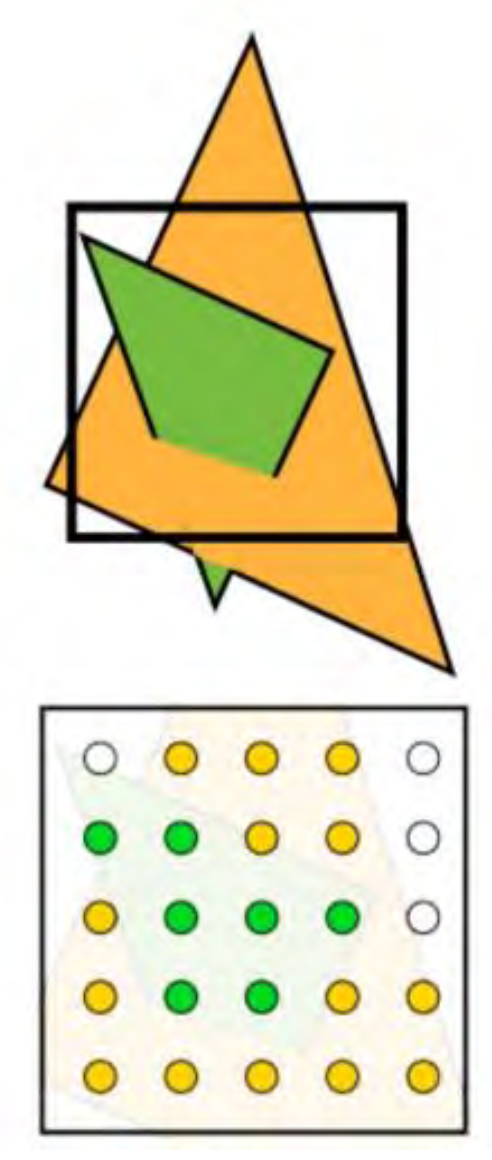
** Interesting read of barycentric coordinates for n-gons: https://www.inf.usi.ch/hormann/barycentric/

Coverage via Samples

- Sample : Discrete measurement of a signal
  - Multisampling vs Supersampling

- Approximate the coverage of the area of a pixel by taking *n* samples
  - Per sample coverage & depth test + texture lookup + alpha blending





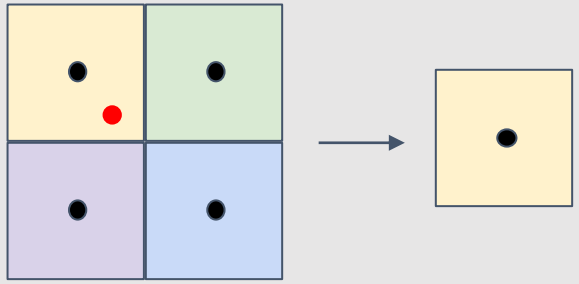
Nearest Neighbor Sampling

- **Idea:** Grab texel nearest to requested location in texture

- **Requires:**
  - 1 memory lookup
  - 0 linear interpolations

$$x' \leftarrow round(x - 0.5), \qquad y' \leftarrow round(y) - 0.5$$

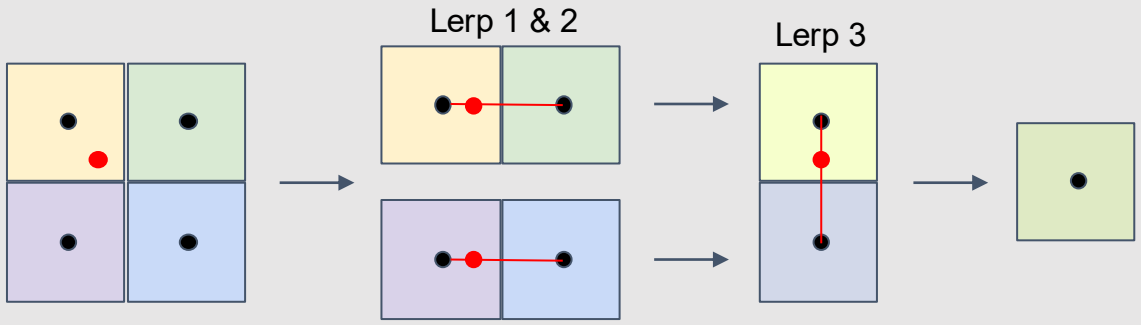
$$t \leftarrow tex.lookup(x', y')$$



Bilinear Interpolation Sampling

- **Idea:** Grab nearest 4 texels and blend them together based on their inverse distance from the requested location
  - Blend two sets of pixels along one axis, then blend the remaining pixels

- **Requires:**
  - 4 memory lookup
  - 3 linear interpolations



$$x' \leftarrow floor(x - 0.5), \qquad y' \leftarrow floor(y - 0.5)$$

$$\Delta x \leftarrow (x - 0.5) - x'$$
$$\Delta y \leftarrow (y - 0.5) - y'$$

$$t_{(x,y)} \leftarrow tex.lookup(x', y')$$
$$t_{(x+1,y)} \leftarrow tex.lookup(x' + 1, y')$$
$$t_{(x,y+1)} \leftarrow tex.lookup(x', y' + 1)$$
$$t_{(x+1,y+1)} \leftarrow tex.lookup(x', +1\, y' + 1)$$

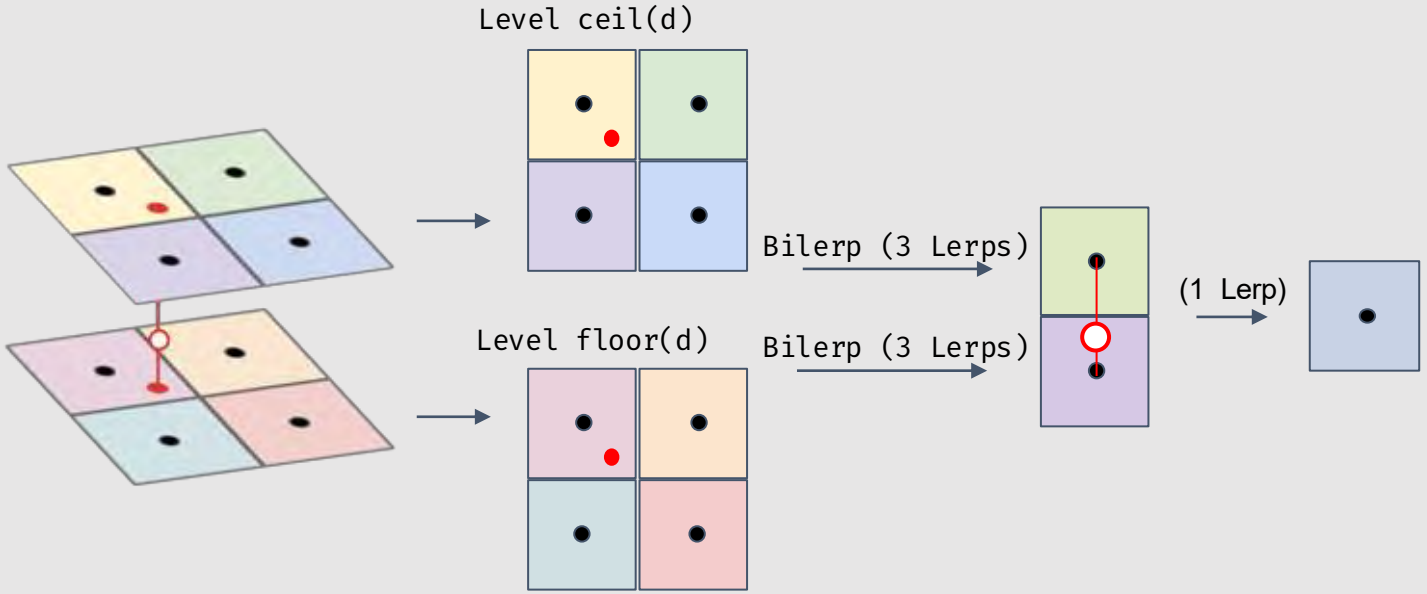
$$t_x \leftarrow (1 - \Delta x) * t_{(x,y)} + \Delta x * t_{(x+1,y)}$$
$$t_y \leftarrow (1 - \Delta x) * t_{(x,y+1)} + \Delta x * t_{(x+1,y+1)}$$

$$t \leftarrow (1 - \Delta y) * t_x + \Delta y * t_y$$

Trilinear Interpolation Sampling

- **Idea:** Perform bilinear interpolation on two layers of the mip-map that represents proper minification/magnification, blending the results together

- **Requires:**
  - 8 memory lookup
  - 7 linear interpolations



$$L_x{}^2 \leftarrow \frac{du}{dx}^2 + \frac{dv}{dx}^2$$
$$L_y{}^2 \leftarrow \frac{du}{dy}^2 + \frac{dv}{dy}^2$$

$$L \leftarrow \sqrt{\max(L_x{}^2, L_y{}^2)}$$
$$d \leftarrow log_2\, L$$

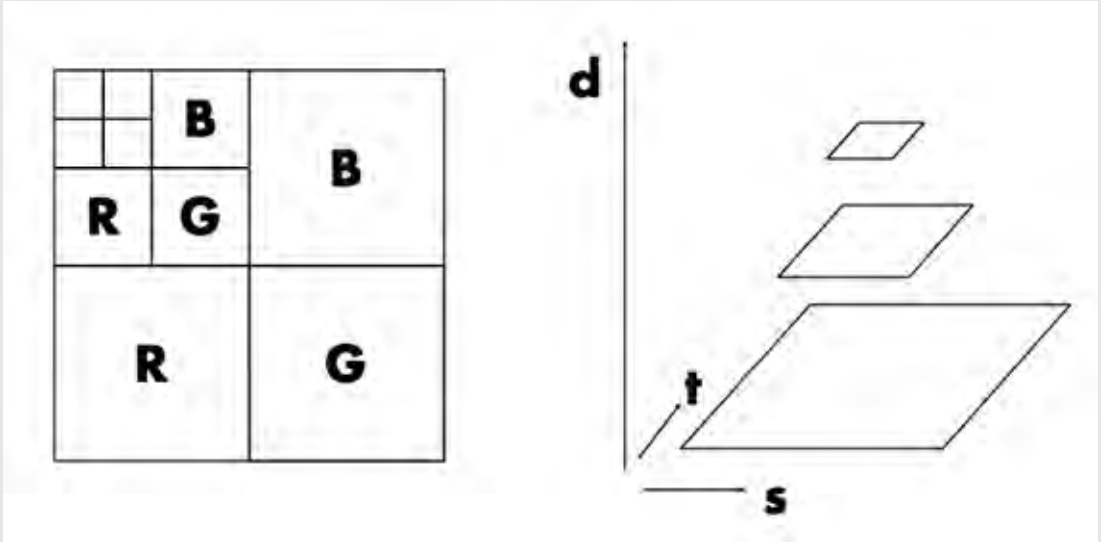
$$d' \leftarrow floor(d)$$
$$\Delta d \leftarrow d - d'$$

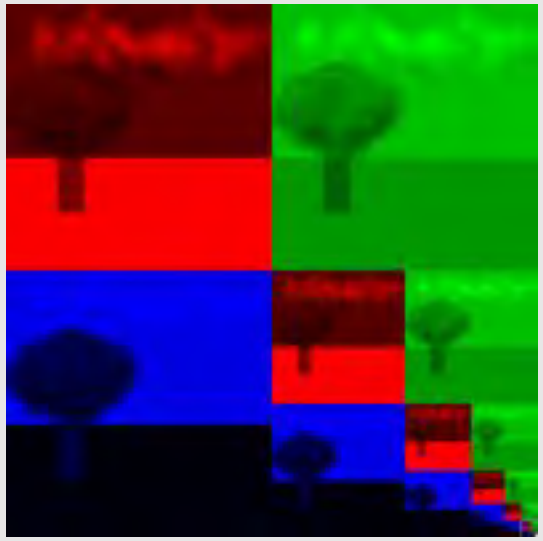
$$t_d \leftarrow tex[d'].bilinear(x, y)$$
$$t_{d+1} \leftarrow tex[d'+1].bilinear(x, y)$$
$$t \leftarrow (1 - \Delta d) * t_d + \Delta d * t_{d+1}$$

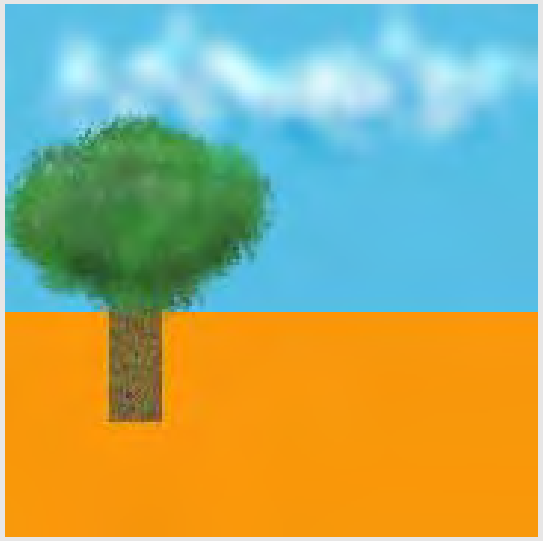
Mip-Map [L. Williams '83]

- Storing an RGB Mip-Map can be fit into an image twice the width and twice the height of the original image
  - See diagram for proof :)
  - Does not work as nicely for RGBA!

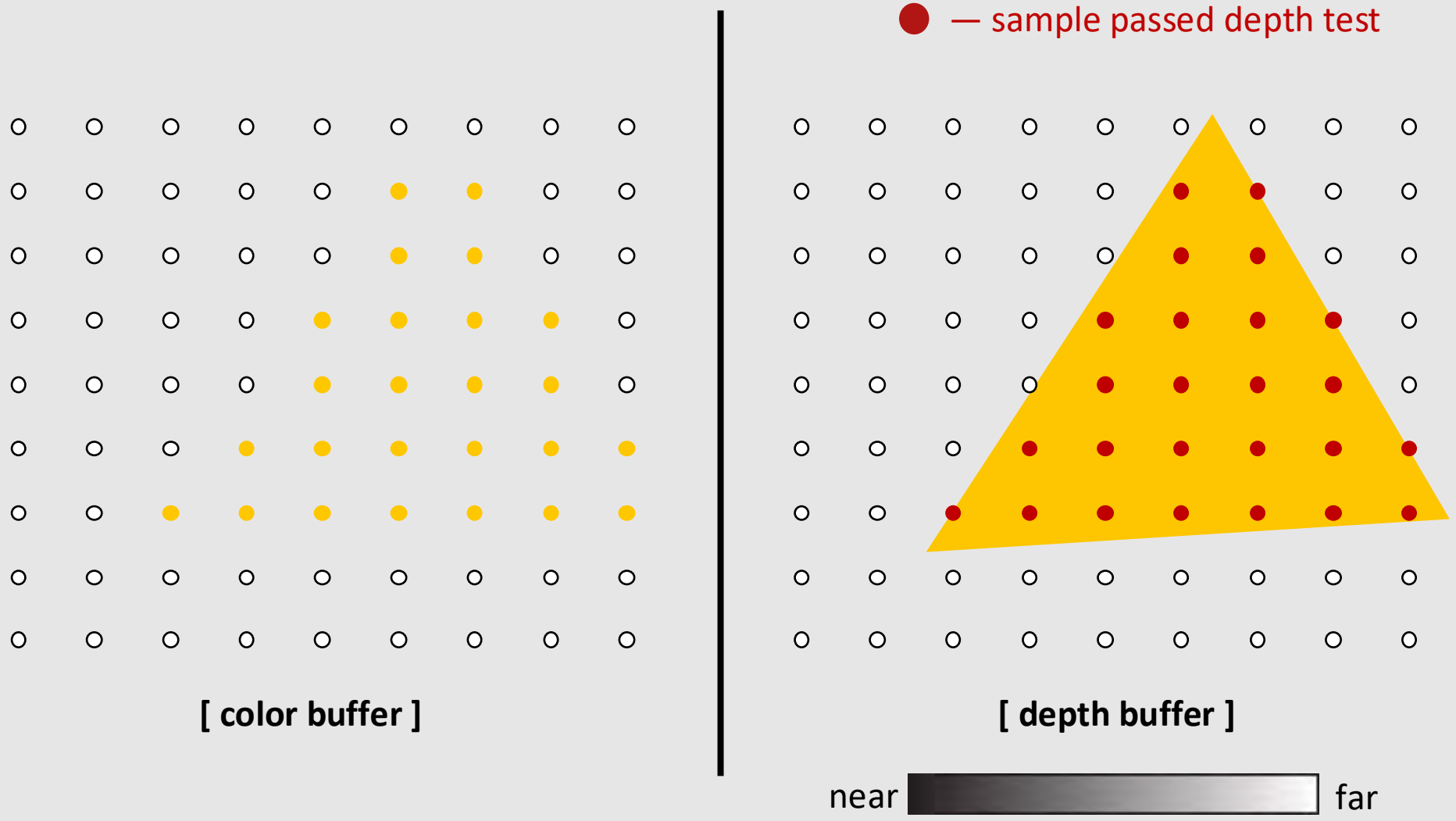
- **Issue:** bad spatial locality
  - Requesting a texel requires lookup in 3 very different regions of an image



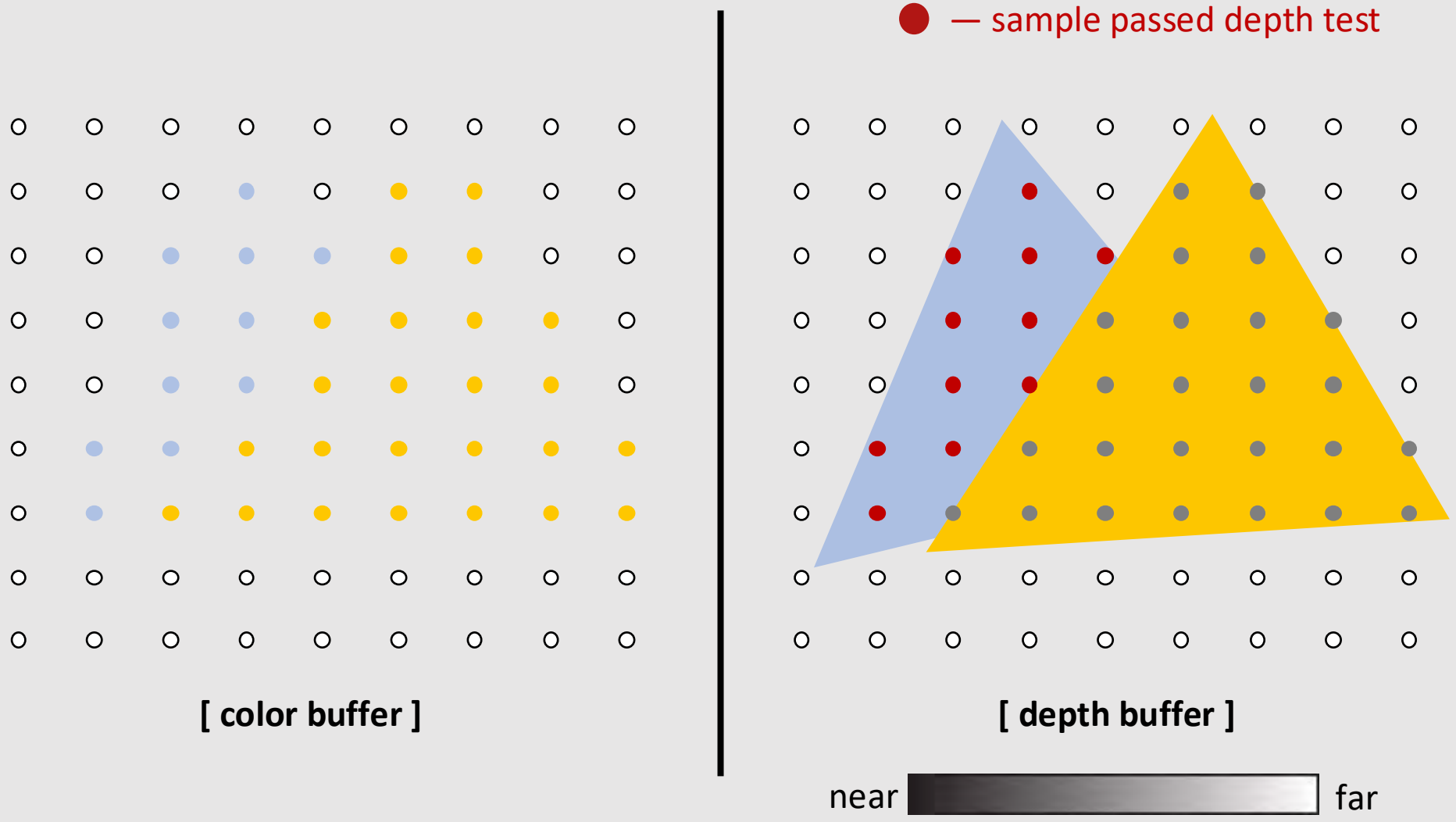




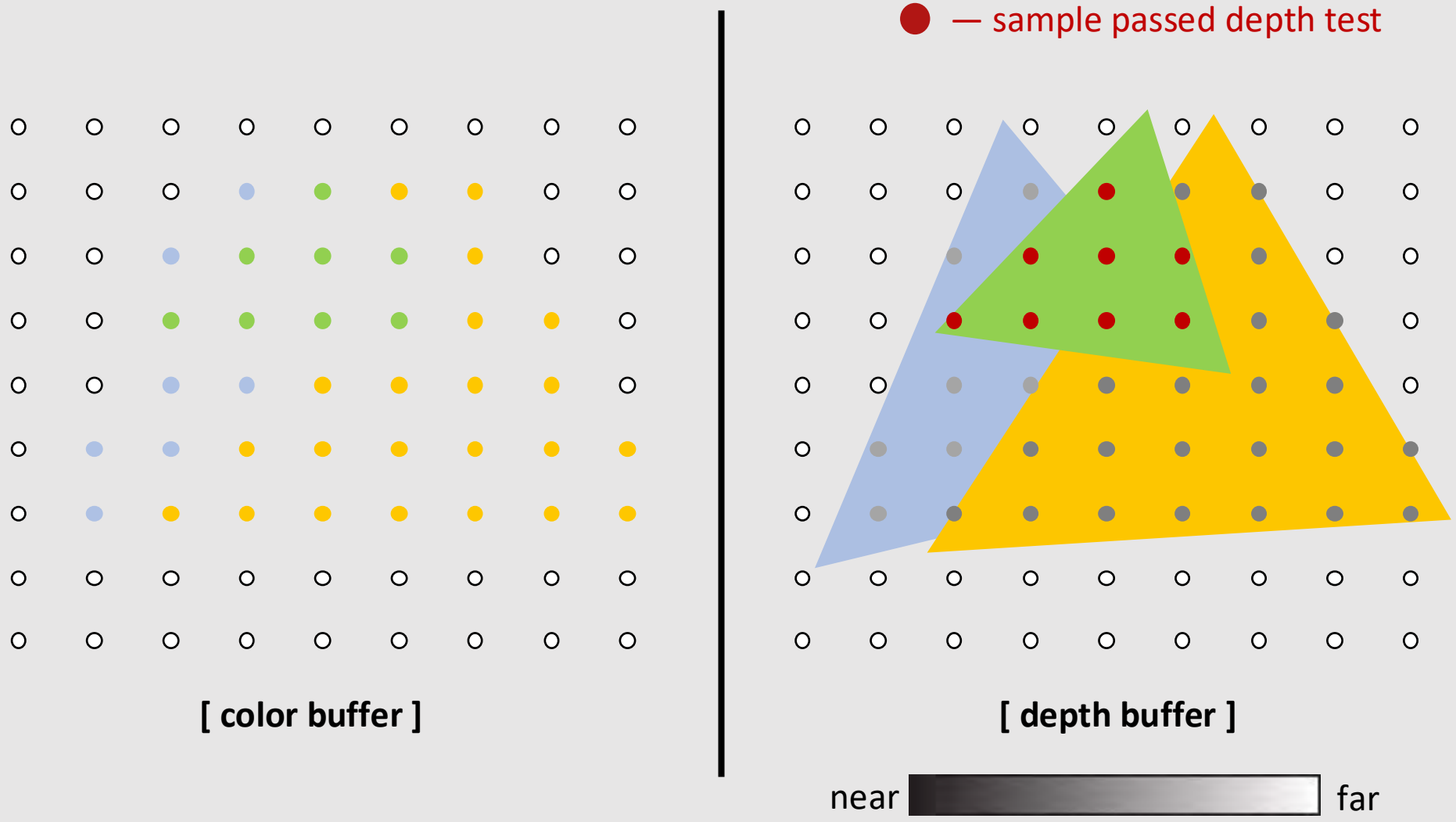
Depth Buffer (Z-buffer)



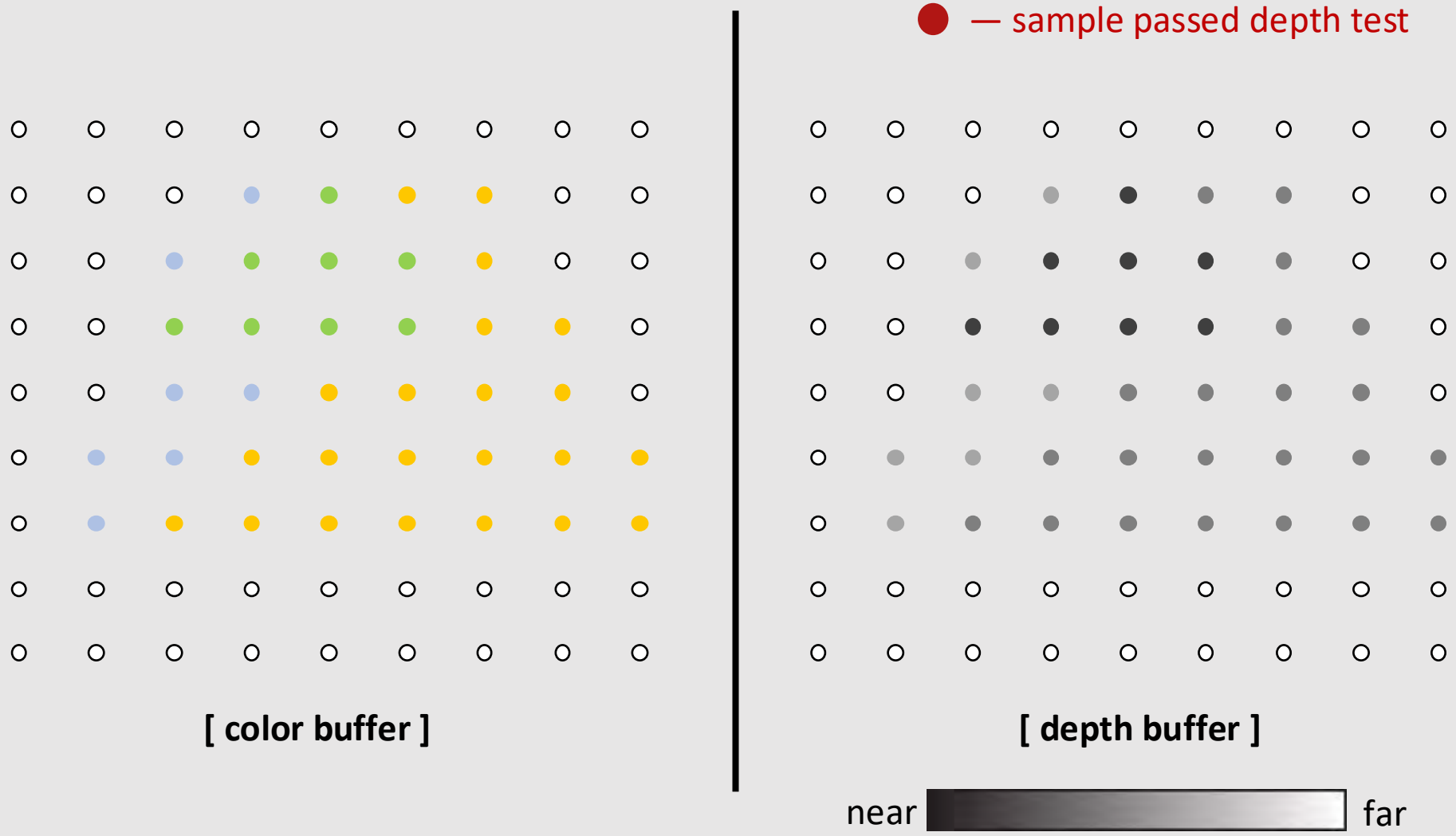
Depth Buffer (Z-buffer)



Depth Buffer (Z-buffer)



Depth Buffer (Z-buffer)



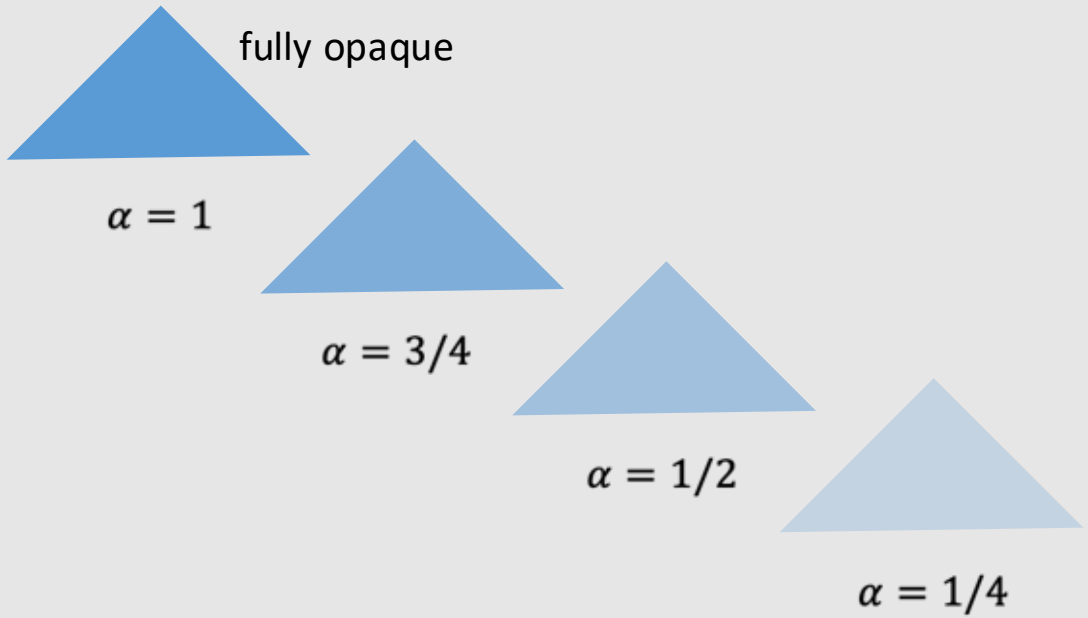
Alpha Values

- Common image format: RGBA
  - Alpha channel specifies ‘opacity’/transparency of object
  - Most common encoding is 8-bits per channel

- Compositing A over B != B over A
  - Consider the extreme case of two opaque objects…

- Non-premultiplied alpha vs Premultiplied alpha

$$C = \alpha_B B + (1 - \alpha_B)\alpha_A A \qquad C' = B' + (1 - \alpha_B)A'$$



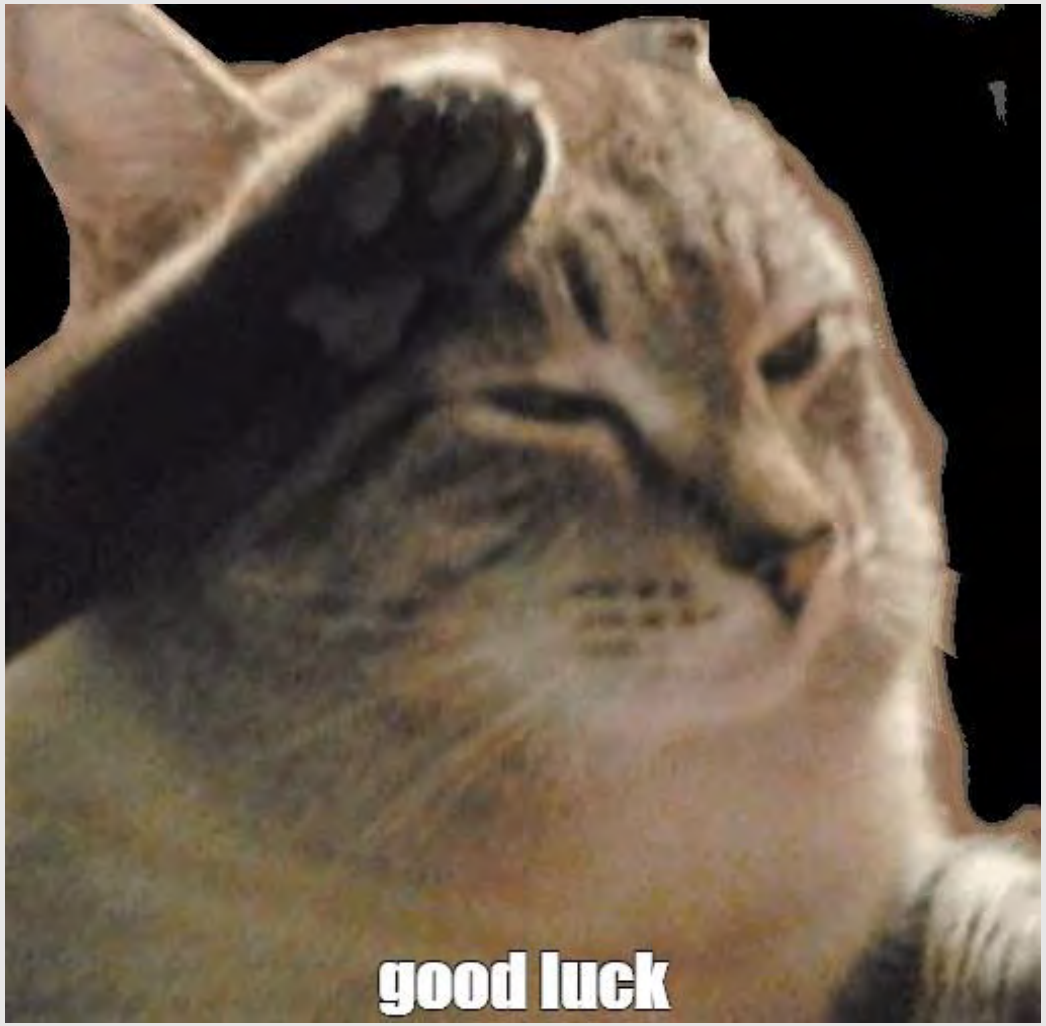
[koala over nyc]

[nyc over…koala?]

$\alpha = 0$

fully transparent

where is the koala…


good luck