Interactive Computer Graphics: Lecture 8

Texture mapping

Some slides adopted from
H. Pfister, Harvard

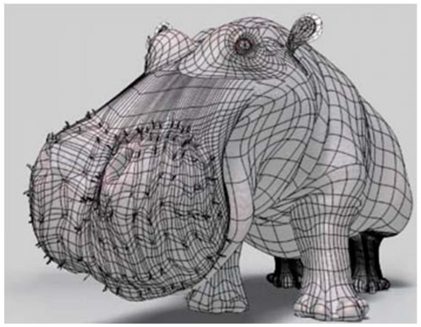
The Problem:

- We don't want to represent all this detail with geometry

The Solution: Textures

- The visual appearance of a graphics scene can be greatly enhanced by the use of texture.
- Consider a brick building, using a polygon for every brick require a huge effort in scene design.
- So why not use one polygon and draw a repeating brick pattern (texture) onto it?

The Quest for Visual Realism



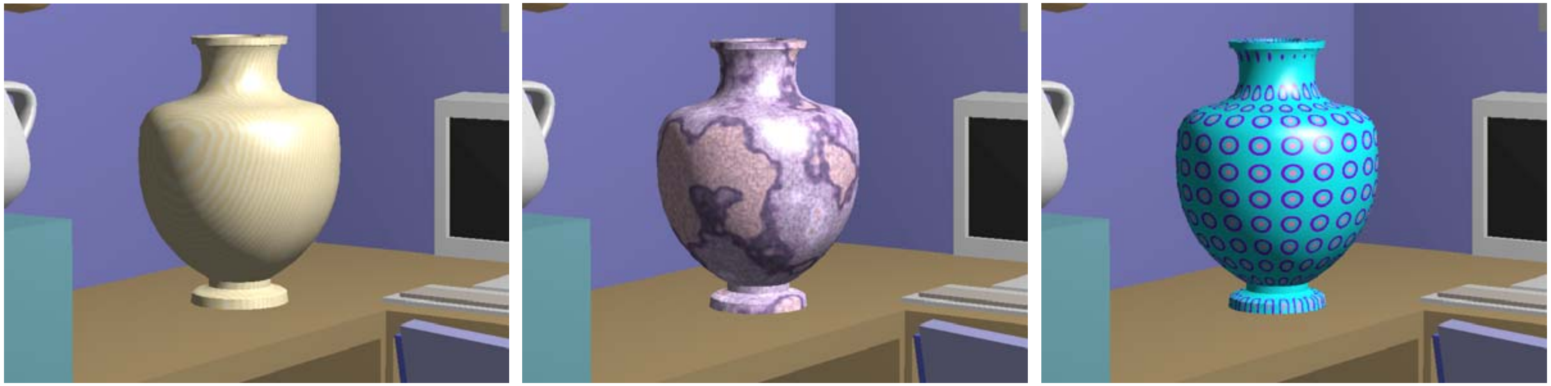
Graphics Lecture 8: Slide 4

Texture Definition

- Textures may be defined as:
  - One-dimensional functions
    - parameter can have arbitrary domain (e.g., incident angle)
  - Two-dimensional functions
    - information is calculated for every (u, v), many possibilities
  - Raster images ("texels")
    - Most often used method
    - Images from scanner, photos or calculation
  - Three-dimensional functions
    - Volume $T(u, v, w)$
- Procedural texture vs. raster data

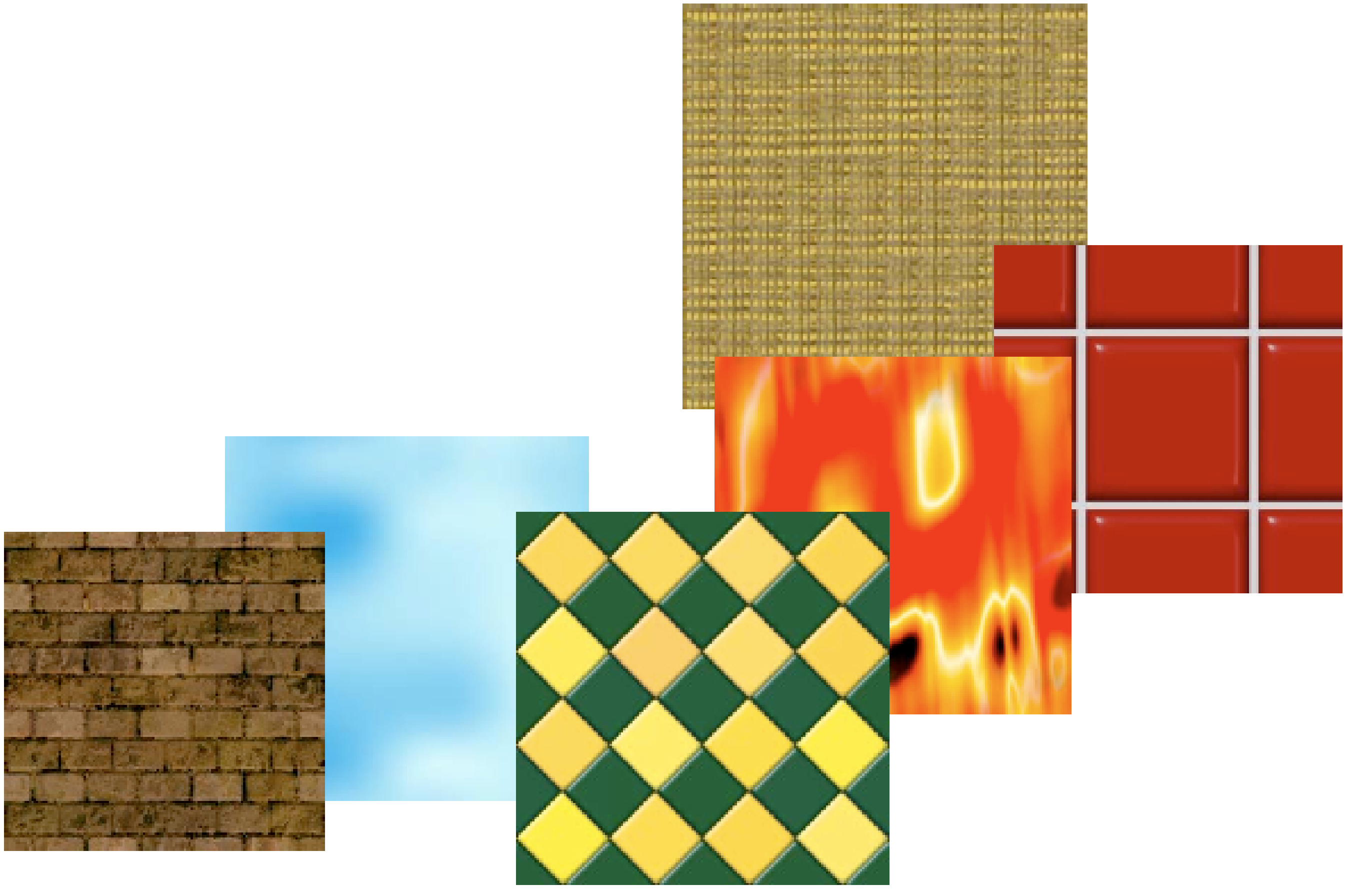
Procedural textures

- Write a function: $F(\mathbf{p}) \rightarrow \text{color}$

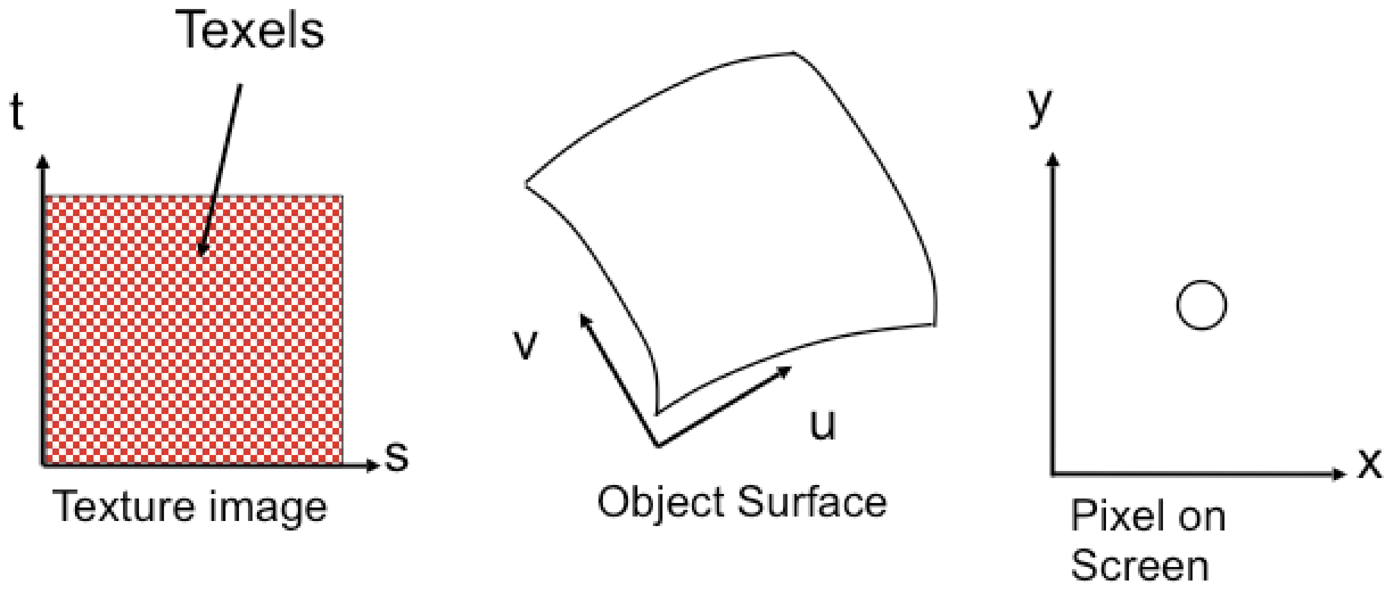


- non-intuitive
- difficult to match a texture that already exists in the ‘real’ world

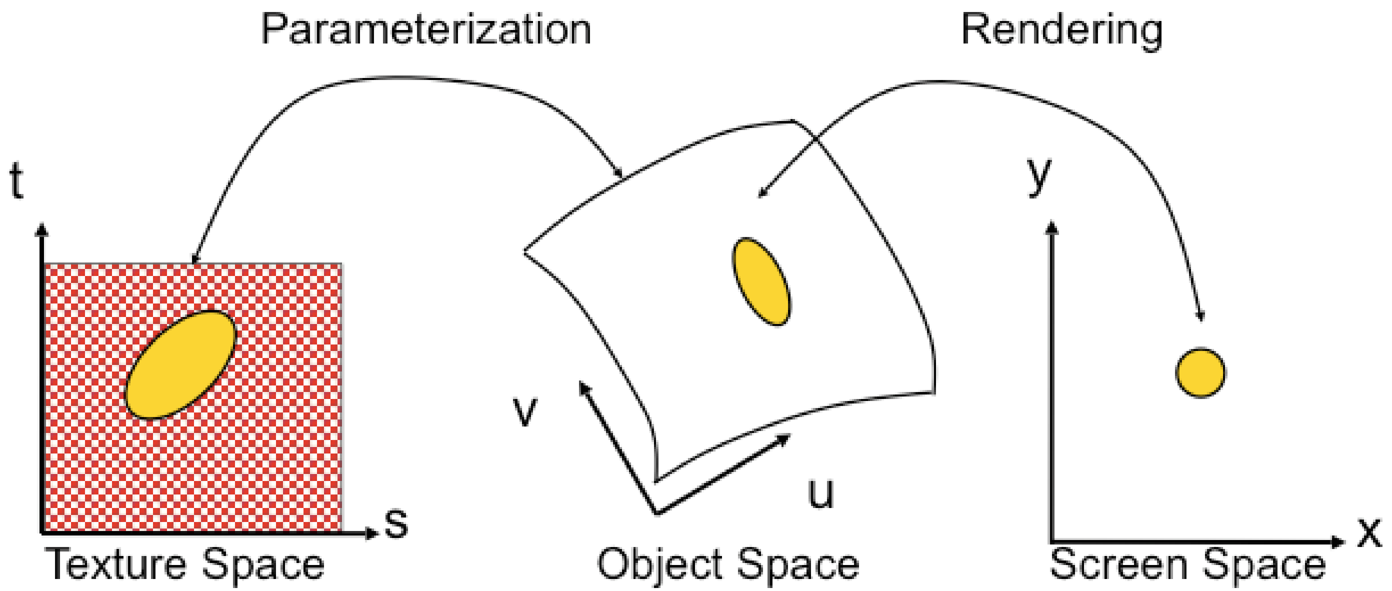
Photo textures



The concept of texture mapping

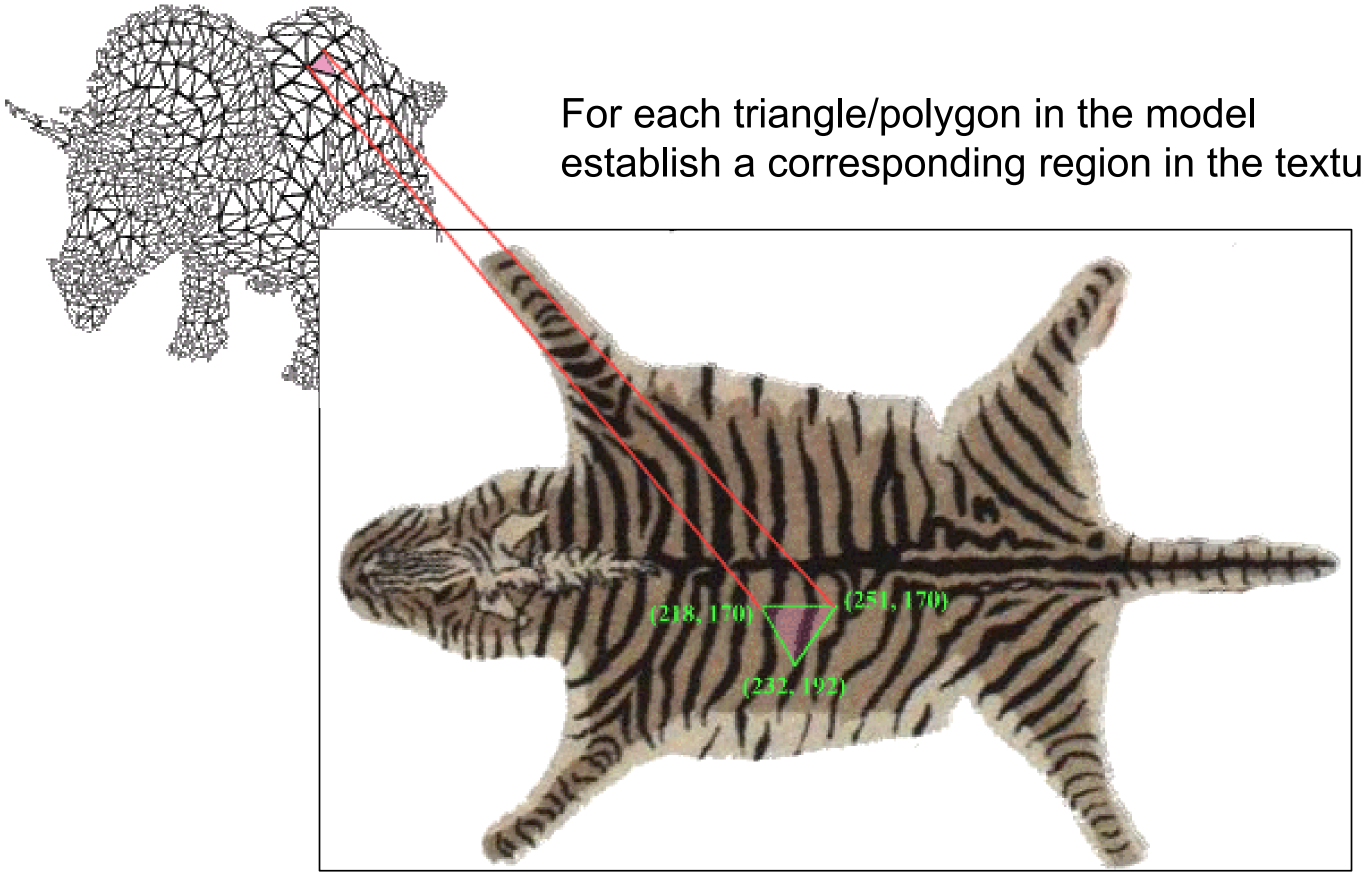


Texture mapping: Terminology



The concept of texture mapping

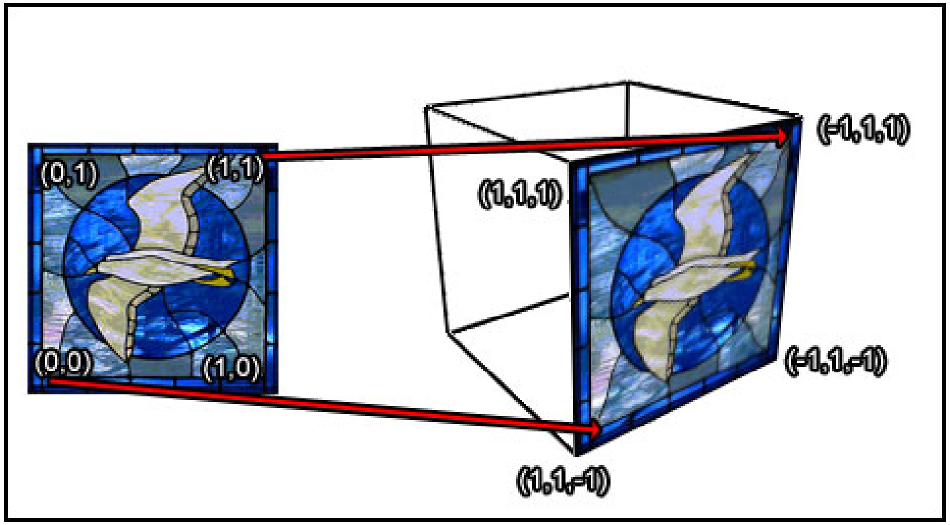
For each triangle/polygon in the model establish a corresponding region in the texture



During rasterization interpolate the coordinate indices into the texture map

Parametrization

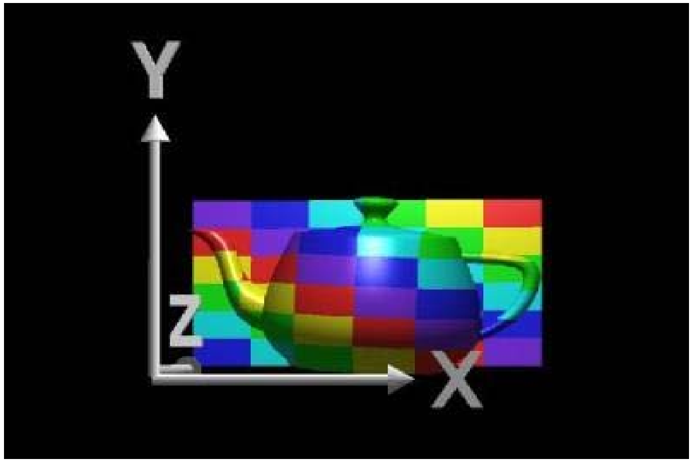
- How to do the mapping?

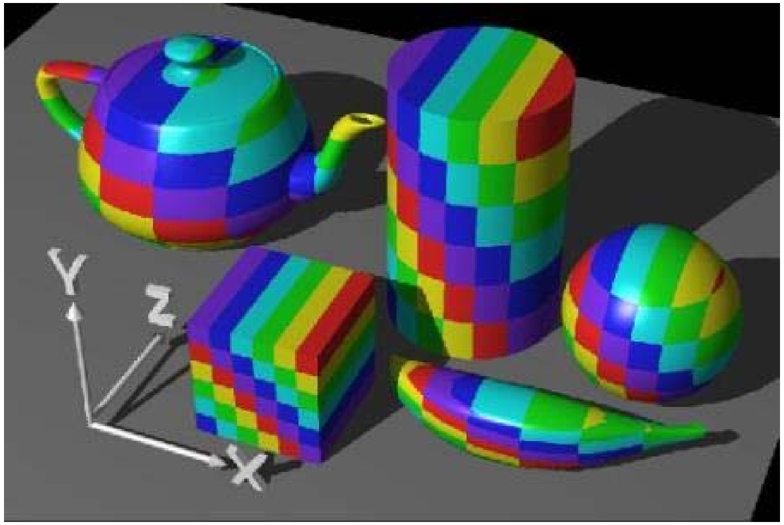


- Usually objects are not that simple

Parametrization: Planar

- Planar mapping: dump one of the coordinates

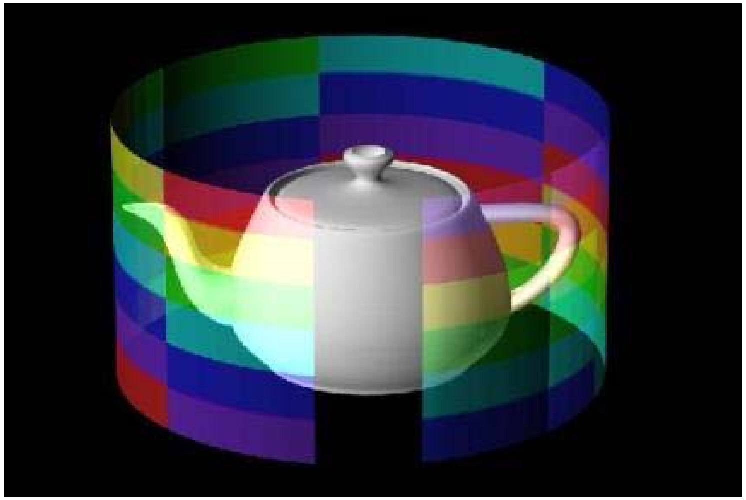


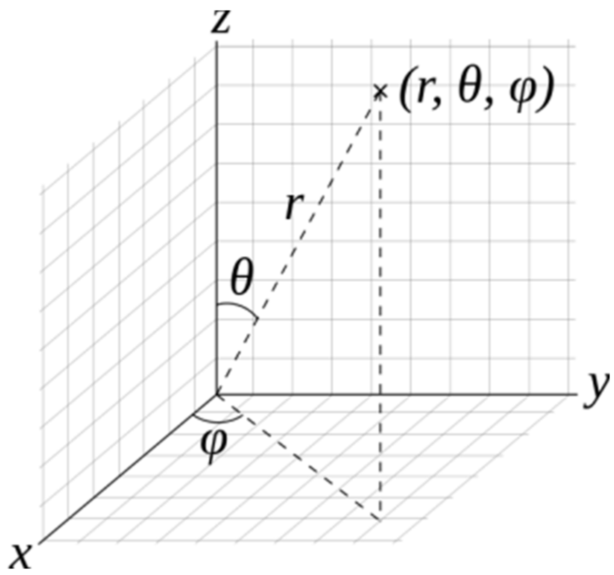


Only looks good from the front!

Parametrization

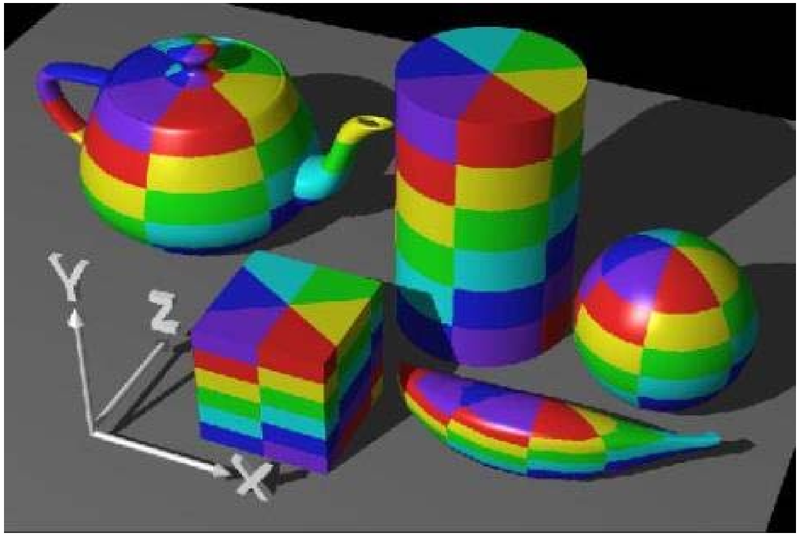
- Cylindrical and spherical mapping: compute angles between vertex and object center
- Compare to polar/spherical coordinate systems

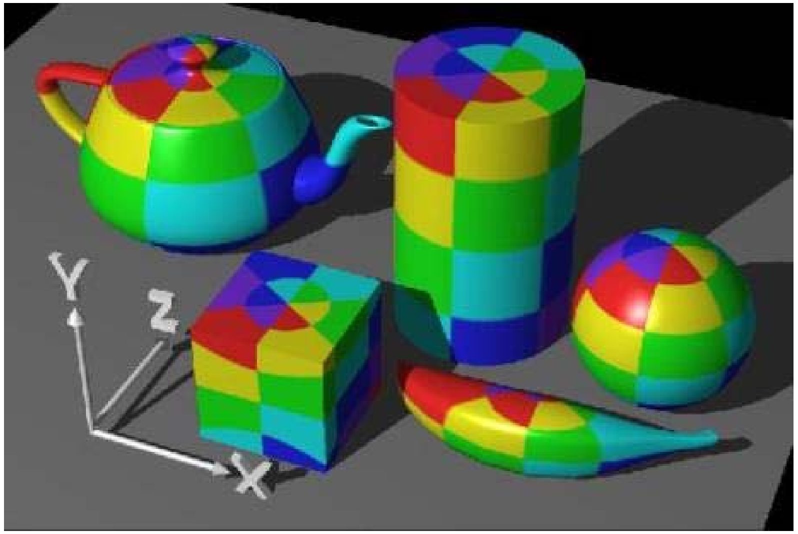




Parametrization

- Cylindrical and spherical mapping



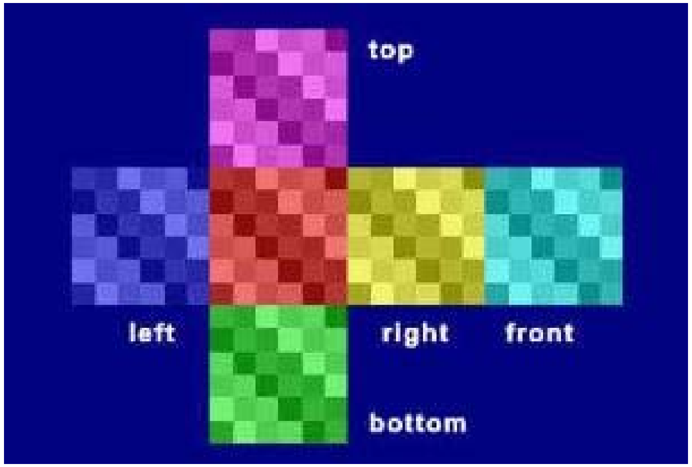


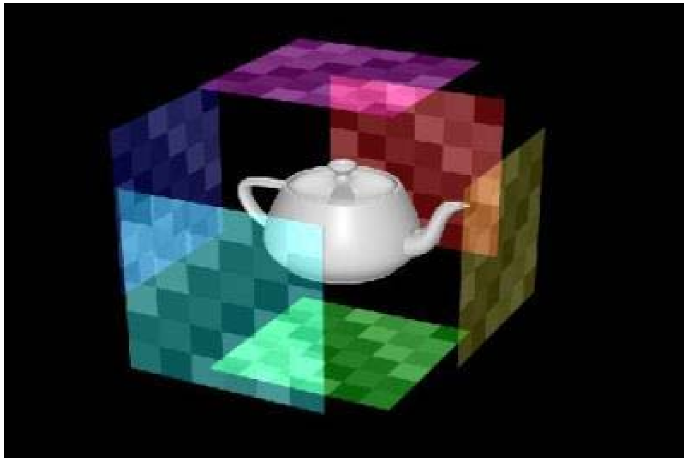
Cylindrical

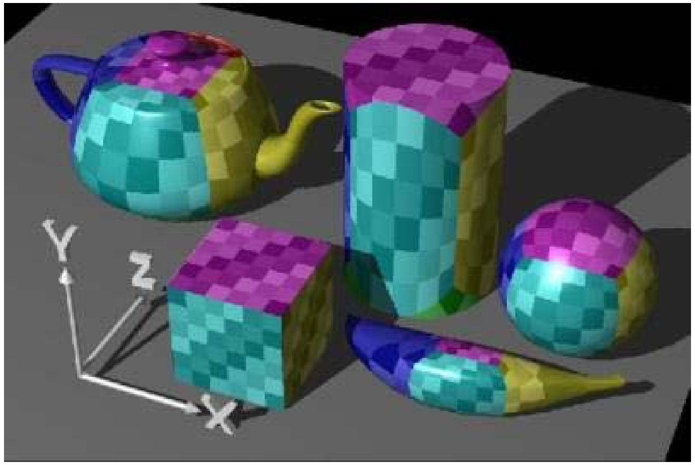
Spherical

Parametrization: Box

- Box mapping: used mainly for environment mapping (see later)

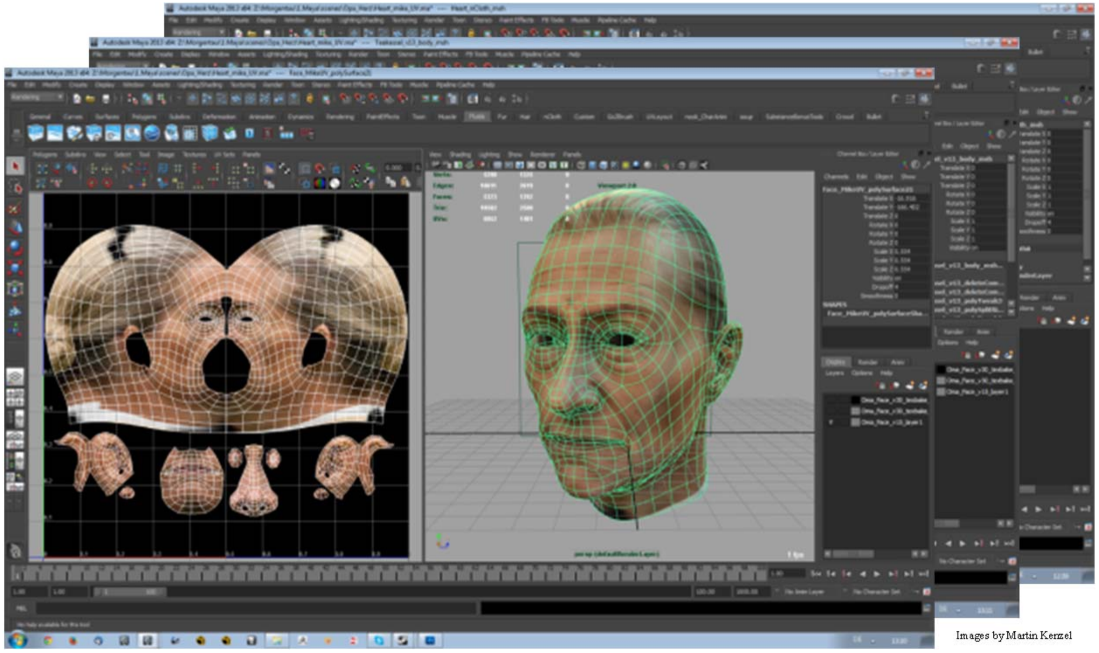






Parametrization

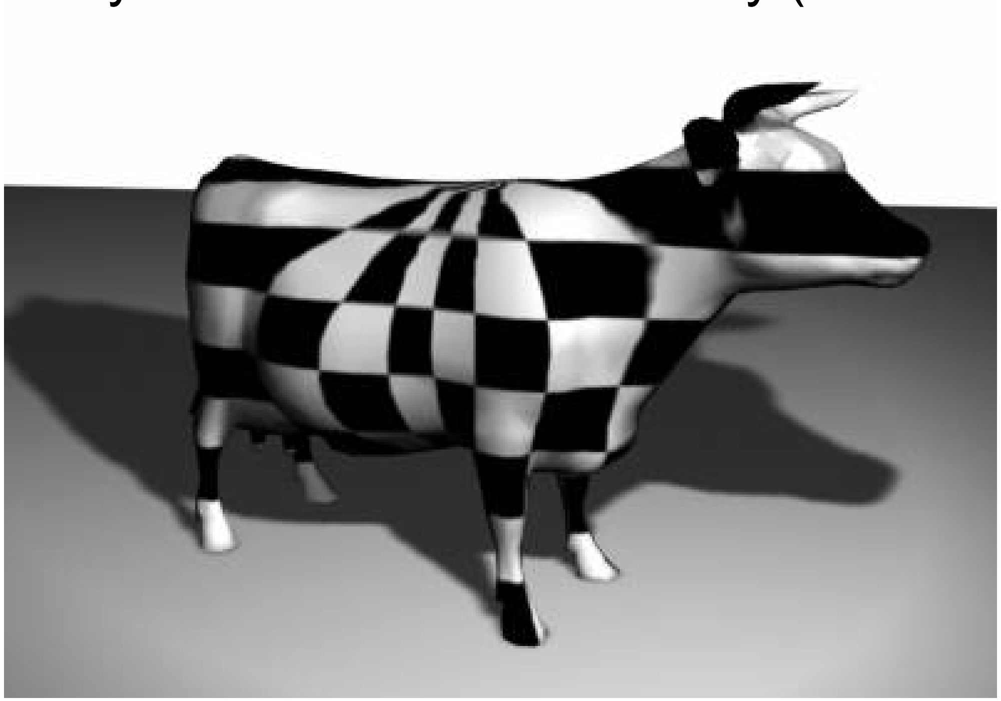
- Manual mapping using CAD Software
  - "Unwrapping"



Images by Martin Kerzel

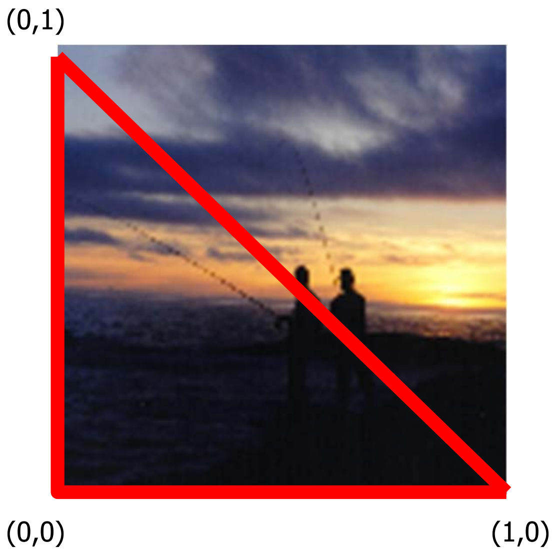
Parameterization problems

- All mappings have distortions and singularities
- Often they need to be fixed manually (CAD software)



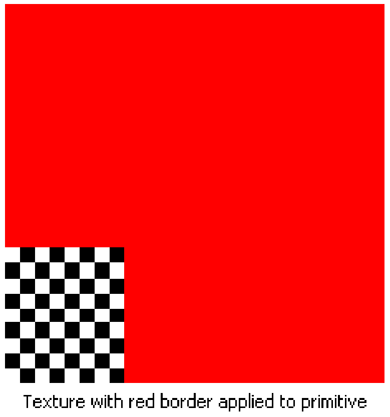
Texture Coordinates

- Specify a texture coordinate at each vertex
- Canonical texture coordinates (0,0) → (1,1)
- Often the texture size is a power of 2 (but it doesn't have to be)
- How can we tile this texture?

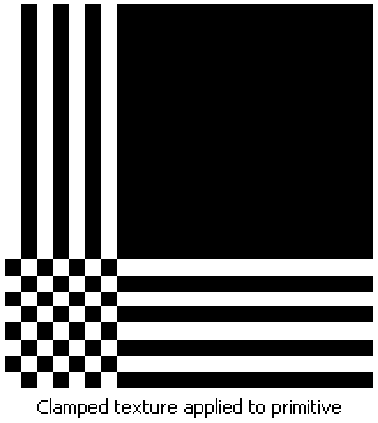


Texture Adressing

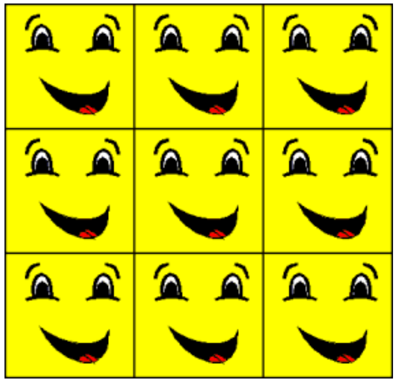
- What happens outside [0,1]?
- Border, repeat, clamp, mirror
- Also called texture addressing modes



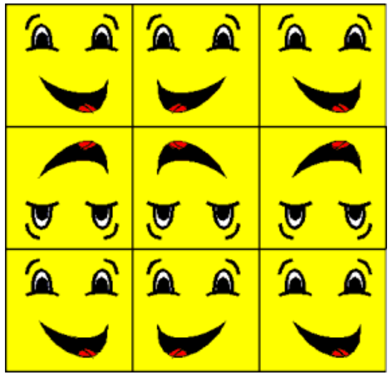
Static color



Clamped

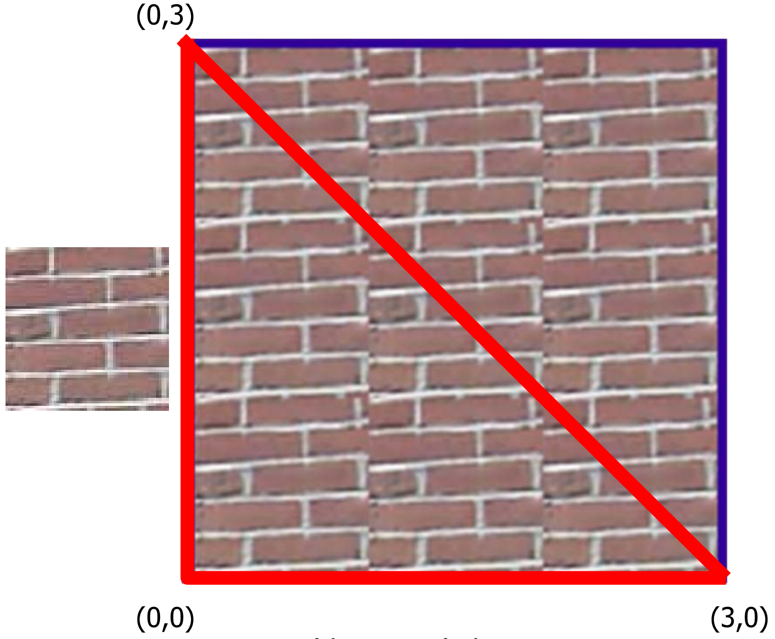


Repeated

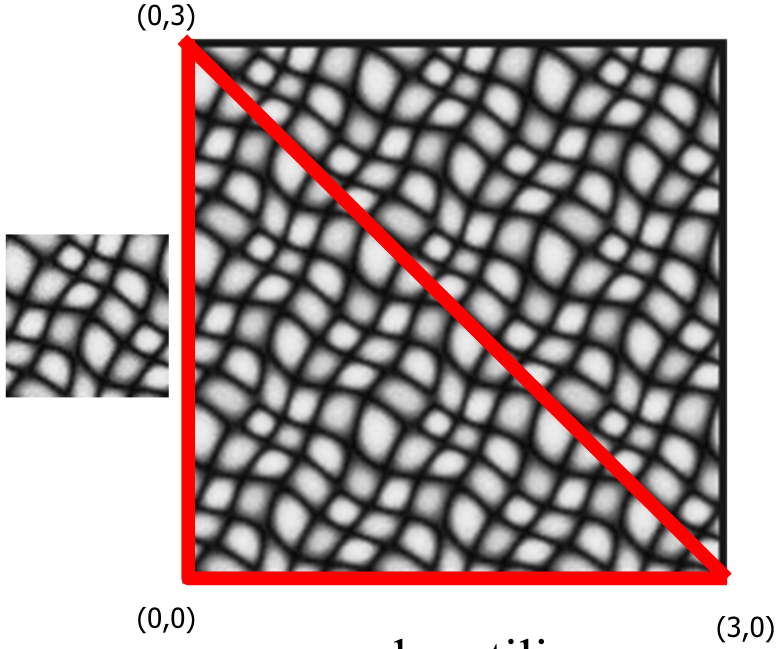


Mirrored

Tiling Texture

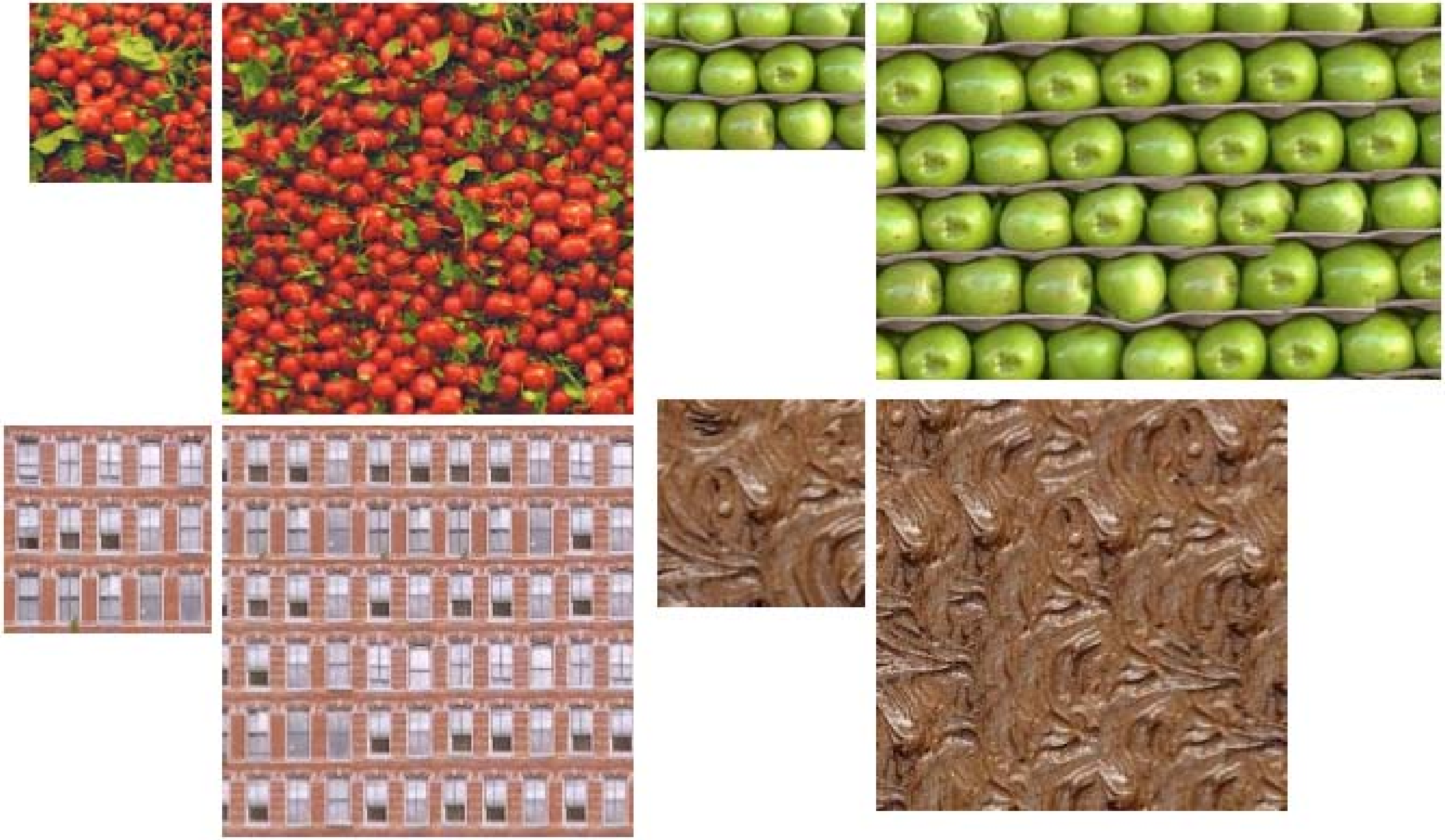


tiles with
visible seams



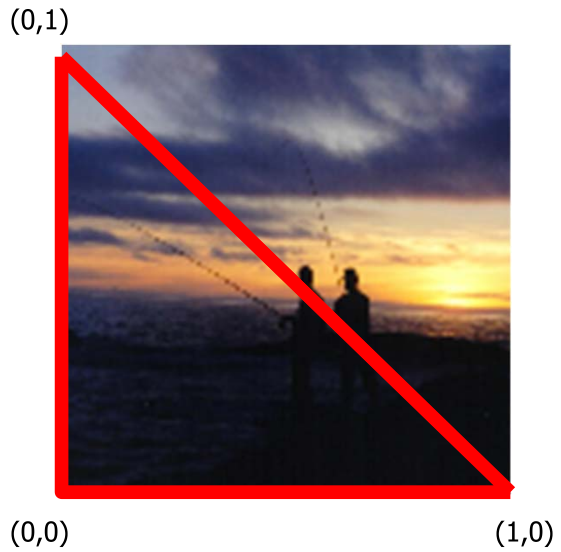
seamless tiling
(repeating)

Texture synthesis



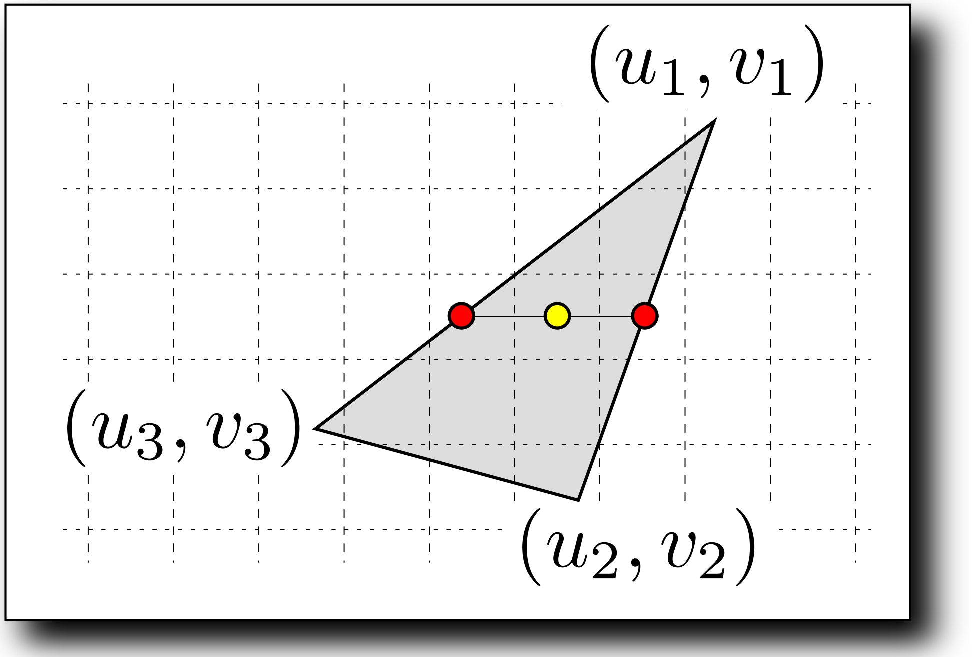
Texture Coordinates

- Specify a texture coordinate at each vertex
- Canonical texture coordinates (0,0) → (1,1)
- Linearly interpolate the values in screen space

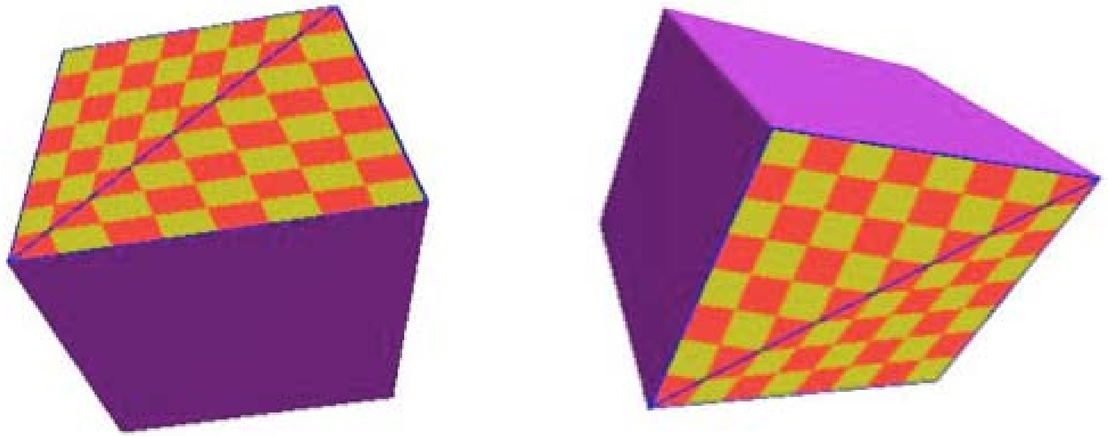


Mapping texture to individual pixels

- Interpolate texture coordinates across scanlines
- Same as Gouraud shading but now for texture coordinates not shading values



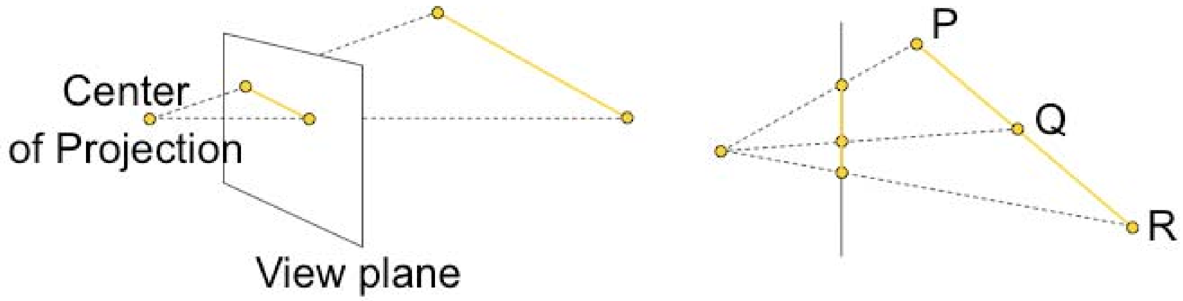
What goes wrong when we linearly interpolate texture coordinates?



- Notice the distortion along the diagonal triangle edge of the cube face after perspective projection

Perspective projection

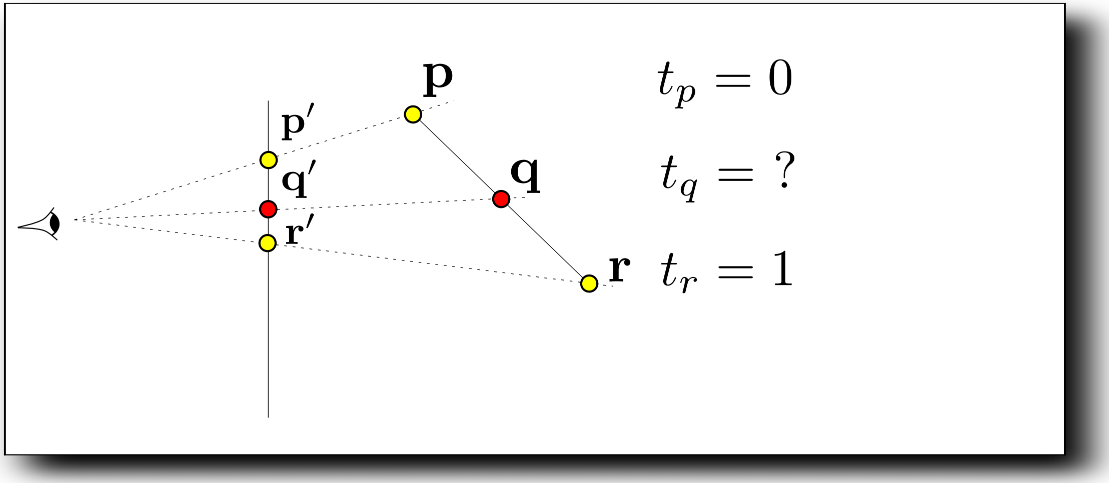
- The problem is that perspective projection does not preserve linear combinations of points!
- In particular; equal distances in 3D space **do not** map to equal distances in screen space



- Linear interpolation in screen space is not the same as linear interpolation in 3D space

How to fix?

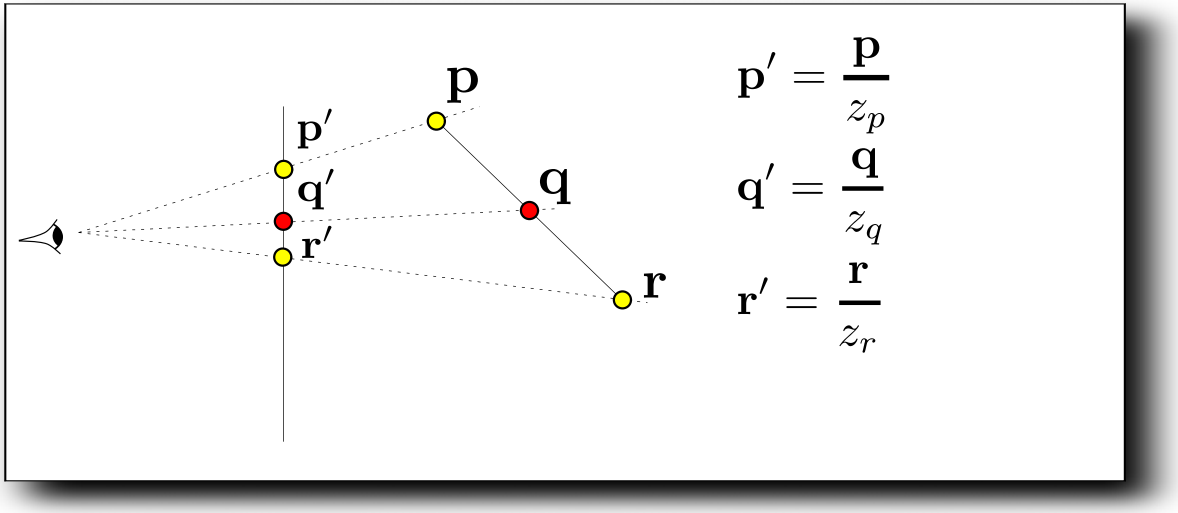
- Assign parameter t to 3-D vertices $\mathbf{p}$ and $\mathbf{r}$
- t controls linear blend of texture coordinates of $\mathbf{p}$ and $\mathbf{r}$
- Let $t = 0$ at $\mathbf{p}$, and $t = 1$ at $\mathbf{r}$
- Assume for simplicity that the image plane is at $z = 1$ *



*i.e. $f = 1$ in the projection matrix

How to fix?

- $\mathbf{p}$ projects to $\mathbf{p}$' and $\mathbf{r}$ projects to $\mathbf{r}$' (simply divide by z coordinate)
- What value should t have at location $\mathbf{q}$'?



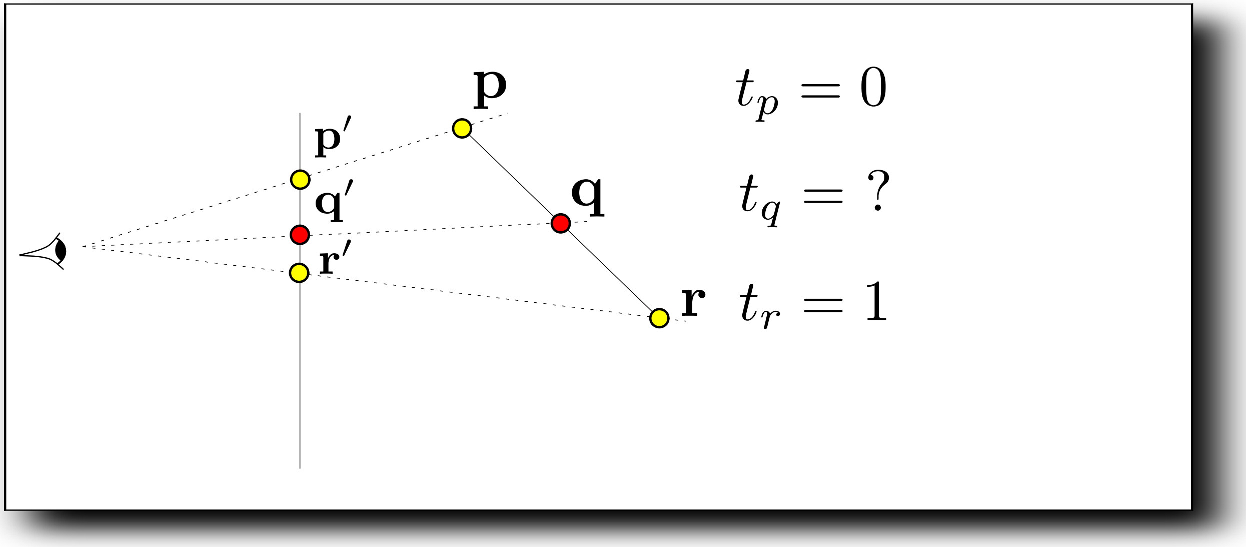
$$\mathbf{p}' = \frac{\mathbf{p}}{z_p}$$

$$\mathbf{q}' = \frac{\mathbf{q}}{z_q}$$

$$\mathbf{r}' = \frac{\mathbf{r}}{z_r}$$

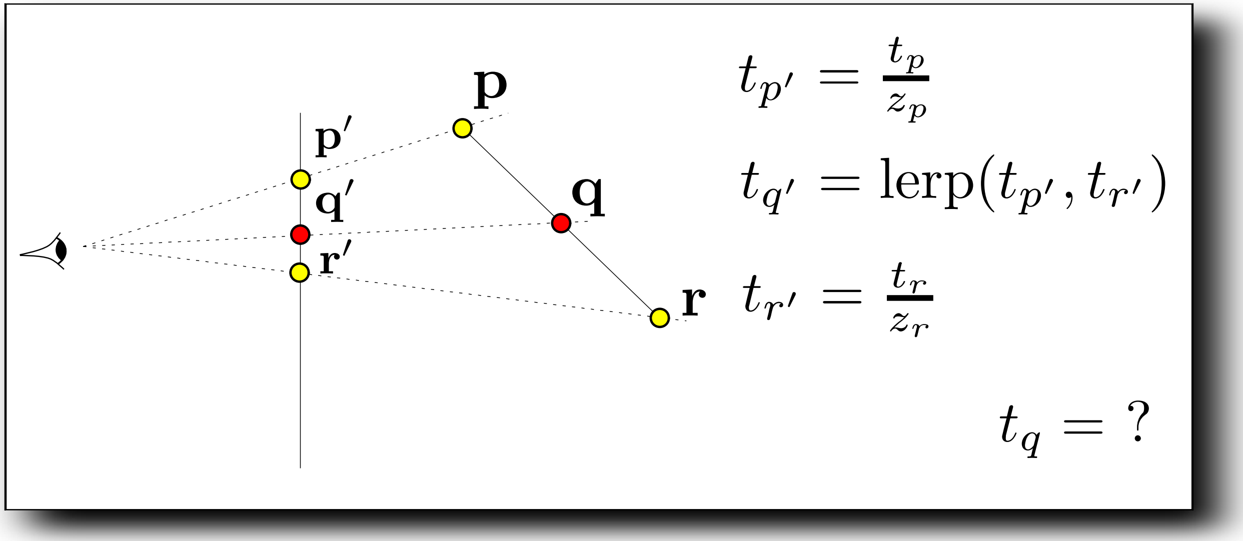
How to fix?

- We cannot linearly interpolate t between $\mathbf{p}'$ and $\mathbf{r}'$
- Only projected values can be linearly interpolated in screen space
- Solution: perspective-correct interpolation



Perspective Correct Interpolation

- Linearly interpolate t / z (not t) between **p'** and **r'**.
  - Compute $t_{\mathbf{p}'} = t_{\mathbf{p}} / z_{\mathbf{p}}$ $\quad t_{\mathbf{r}'} = t_{\mathbf{r}} / z_{\mathbf{r}}$
  - Linearly interpolate (lerp) $t_{\mathbf{p}'}$ and $t_{\mathbf{r}'}$ to get $t_{\mathbf{q}'}$ at location **q**'
- But, we want the un-projected parameter $t_{\mathbf{q}}$ (not $t_{\mathbf{q}'}$)



$$t_{p'} = \frac{t_p}{z_p}$$

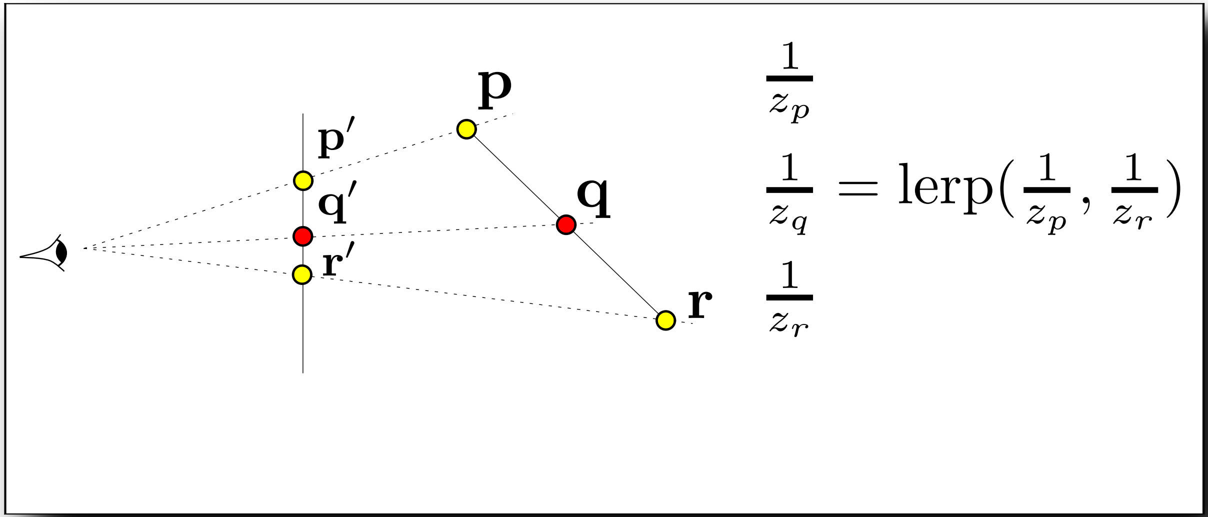
$$t_{q'} = \mathrm{lerp}(t_{p'}, t_{r'})$$

$$t_{r'} = \frac{t_r}{z_r}$$

$$t_q = ?$$

Perspective Correct Interpolation

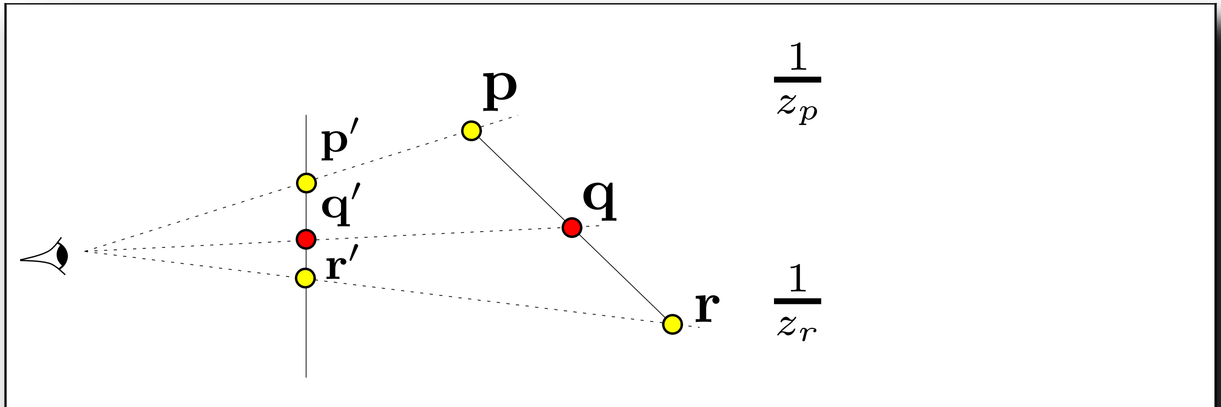
- The parameters $t_{\mathbf{p}'}$ & $t_{\mathbf{q}'}$ are related to $t_{\mathbf{p}}$ & $t_{\mathbf{q}}$ by perspective factors of $1/z_{\mathbf{p}}$ and $1/z_{\mathbf{q}}$
  - $\mathrm{lerp}\ 1 / z_{\mathbf{p}}$ and $1 / z_{\mathbf{r}}$ to obtain $1 / z_{\mathbf{q}}$ at point $\mathbf{q}$ ‘



$$\frac{1}{z_q} = \mathrm{lerp}(\frac{1}{z_p}, \frac{1}{z_r})$$

Perspective Correct Interpolation

- The parameters $t_{\mathbf{p}'}$ & $t_{\mathbf{q}'}$ are related to $t_{\mathbf{p}}$ & $t_{\mathbf{q}}$ by perspective factors of $1/z_{\mathbf{p}}$ and $1/z_{\mathbf{q}}$
  - lerp $1 / z_{\mathbf{p}}$ and $1 / z_{\mathbf{r}}$ to obtain $1 / z_{\mathbf{q}}$ at point $\mathbf{q}'$
  - Divide $t_{\mathbf{q}'}$ by $1 / z_{\mathbf{q}}$ to get $t_{\mathbf{q}}$



$$t_{\mathbf{q}} = t_{\mathbf{q}'} \times z_{\mathbf{q}} = \mathrm{lerp}(t_{\mathbf{p}'}, t_{\mathbf{r}'}) \div \mathrm{lerp}(\frac{1}{z_{\mathbf{p}}}, \frac{1}{z_{\mathbf{r}}})$$

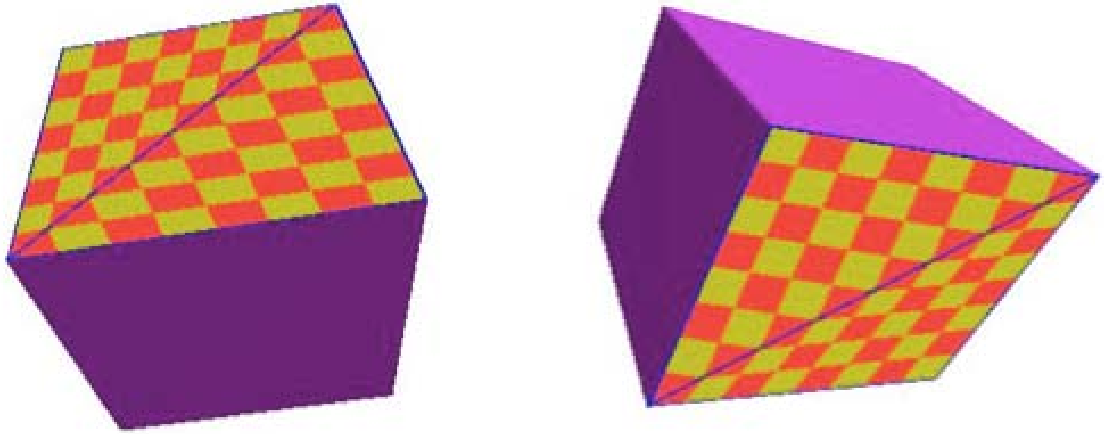
Perspective Correct Interpolation

- Summary:
  - Given texture parameter t at vertices:
    - Compute $1/z$ for each vertex
    - Linearly interpolate $1/z$ across the triangle
    - Linearly interpolate t/z across the triangle
    - Do perspective division:

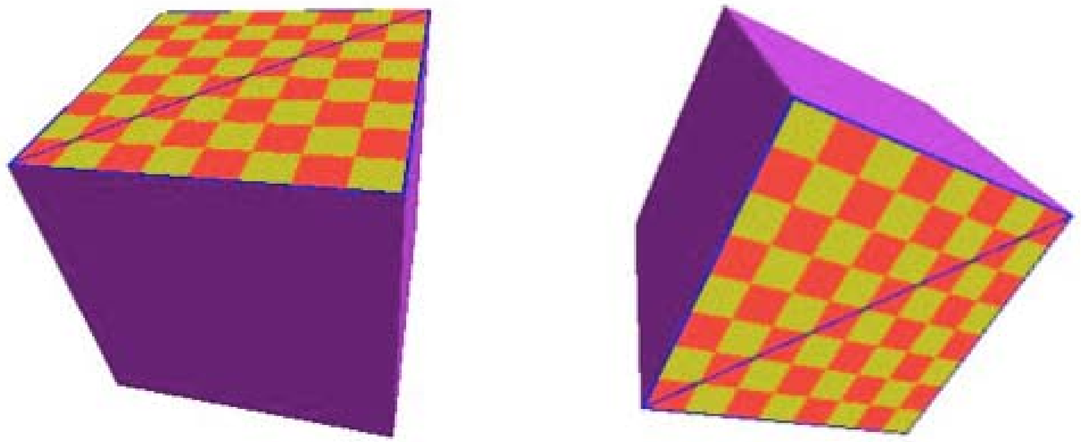
      Divide t/z by $1/z$ to obtain interpolated parameter t

$$t_{\mathbf{q}} = \frac{\mathrm{lerp}\left(\frac{t_{\mathbf{p}}}{z_{\mathbf{p}}}, \frac{t_{\mathbf{r}}}{z_{\mathbf{r}}}\right)}{\mathrm{lerp}\left(\frac{1}{z_{\mathbf{p}}}, \frac{1}{z_{\mathbf{r}}}\right)}$$

What Goes Wrong?

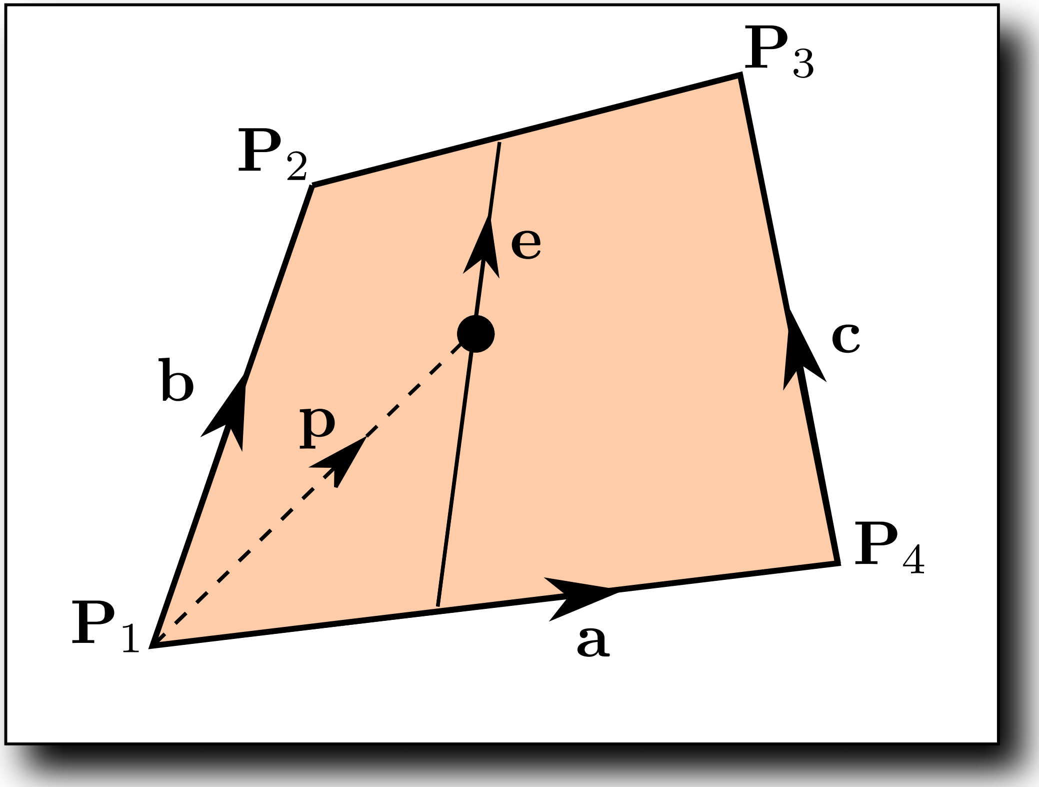


Perspective Correct Interpolation



Mapping texture to individual pixels

Alternative



$\mathbf{P}_{1..4}$: Polygon vertices

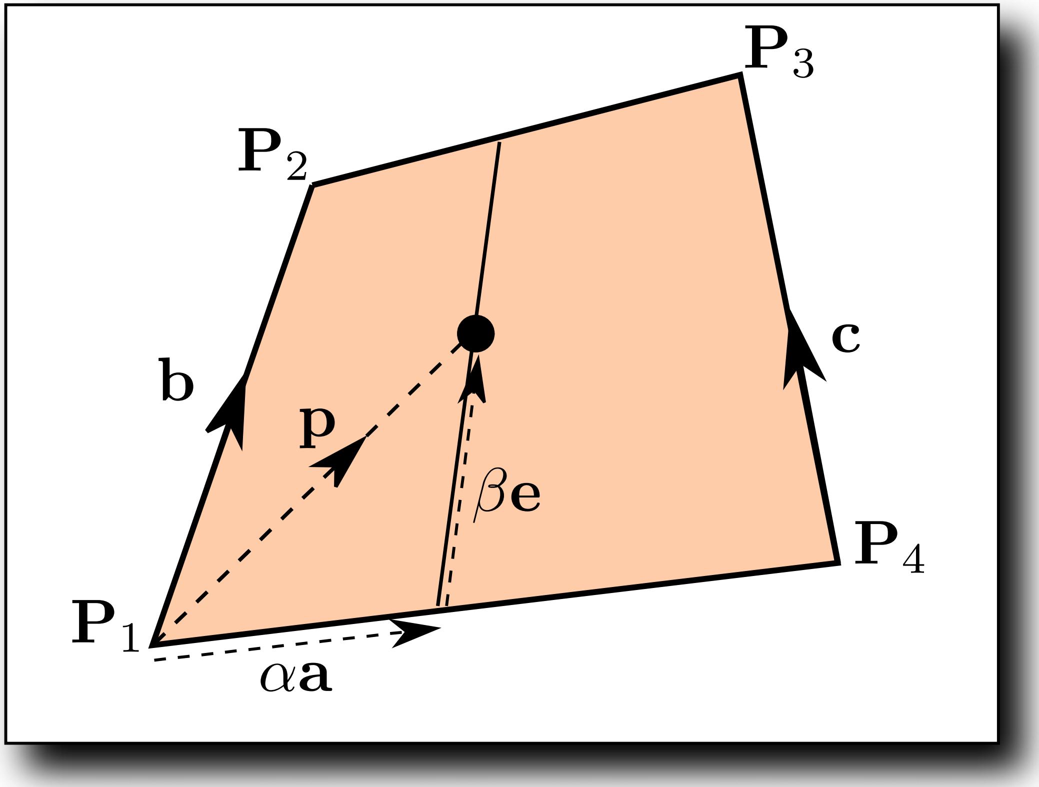
$\mathbf{p}$: Pixel to be textured

Bilinear Texture mapping

Bi-linear Map - Solving for a and b

$\mathbf{p} = \alpha\,\mathbf{a} + \beta\,\mathbf{e}$

$\mathbf{e} = \mathbf{b} + \alpha\,(\mathbf{c} - \mathbf{b})$



so

$\mathbf{p} = \alpha\,\mathbf{a} + \beta\,\mathbf{b} + \alpha\beta\,(\mathbf{c} - \mathbf{b})$ Quadratic in the unknowns !

Non Linearities in texture mapping

- The second order term means that straight lines in the texture may become curved when the texture is mapped.
- However, if the mapping is to a parallelogram:

$$\mathbf{p} = \alpha\,\mathbf{a} + \beta\,\mathbf{b} + \alpha\beta\,(\mathbf{c} - \mathbf{b})$$

and

$$\mathbf{b} = \mathbf{c}$$

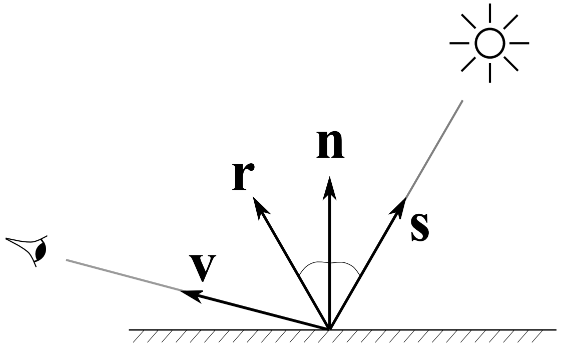
so

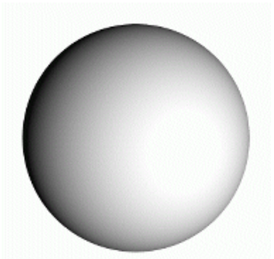
$$\mathbf{p} = \alpha\,\mathbf{a} + \beta\,\mathbf{b}$$

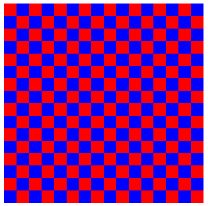
Texture Mapping & Illumination

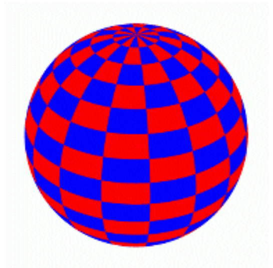
- Texture mapping can be used to alter parts of the illumination equation

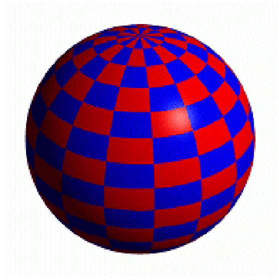
$$L(\omega_r) = k_a I_a + \left(k_d I_d (\mathbf{n}\cdot\mathbf{s}) + k_s I_s (\mathbf{v}\cdot\mathbf{r})^q\right)$$




Constant Diffuse Color

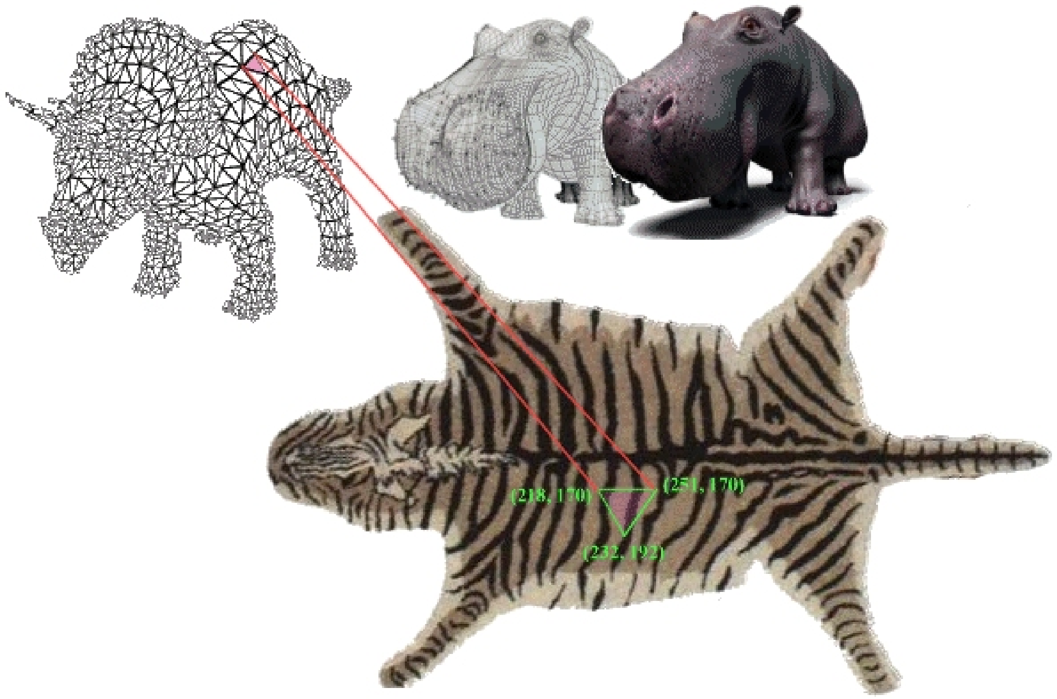

Texture Image


Texture used as Label


Texture used as Diffuse Color

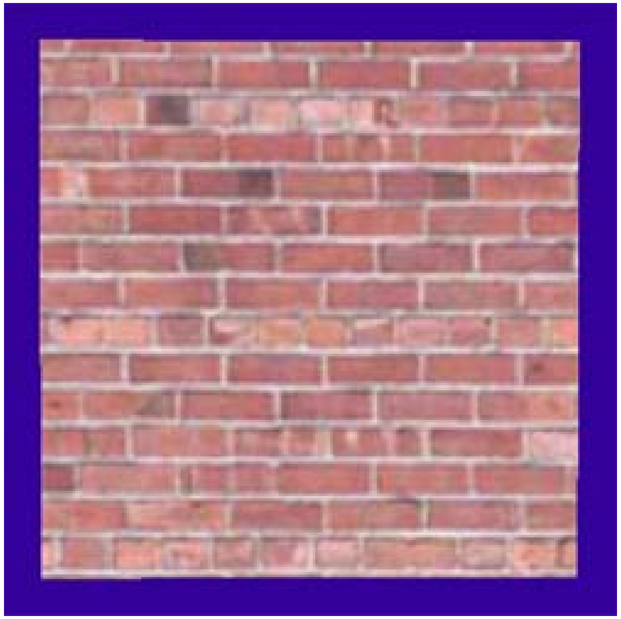
2D Texture Mapping

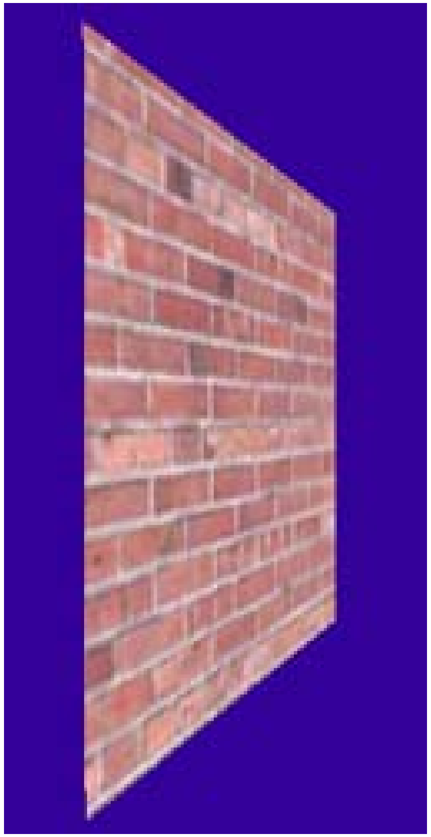
- Increases the apparent complexity of simple geometry
- Requires perspective projection correction
- Can specify variations in shading within a primitive:
  - Illumination
  - Surface Reflectance



What's Missing?

- What's the difference between a real brick wall and a photograph of the wall texture-mapped onto a plane?
- What happens if we change the lighting or the camera position?

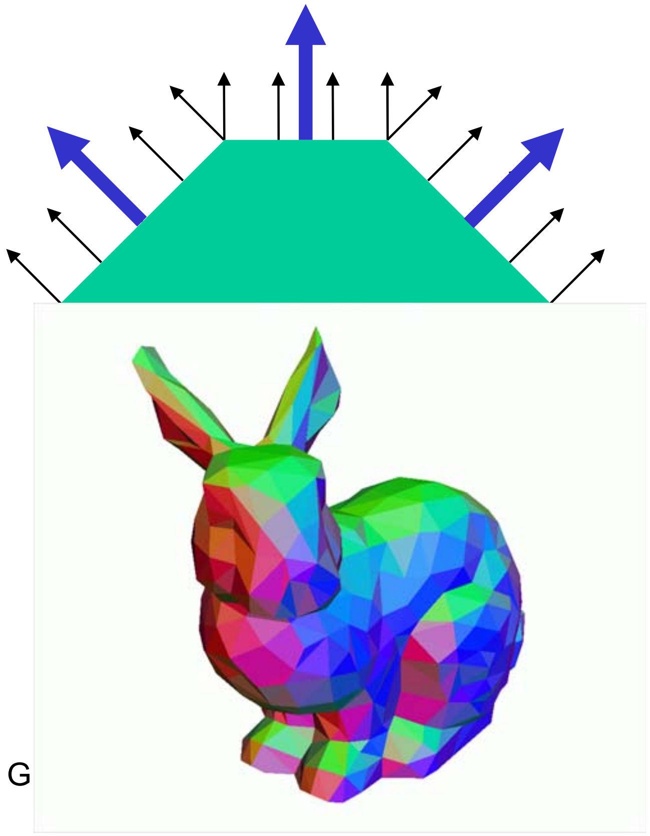


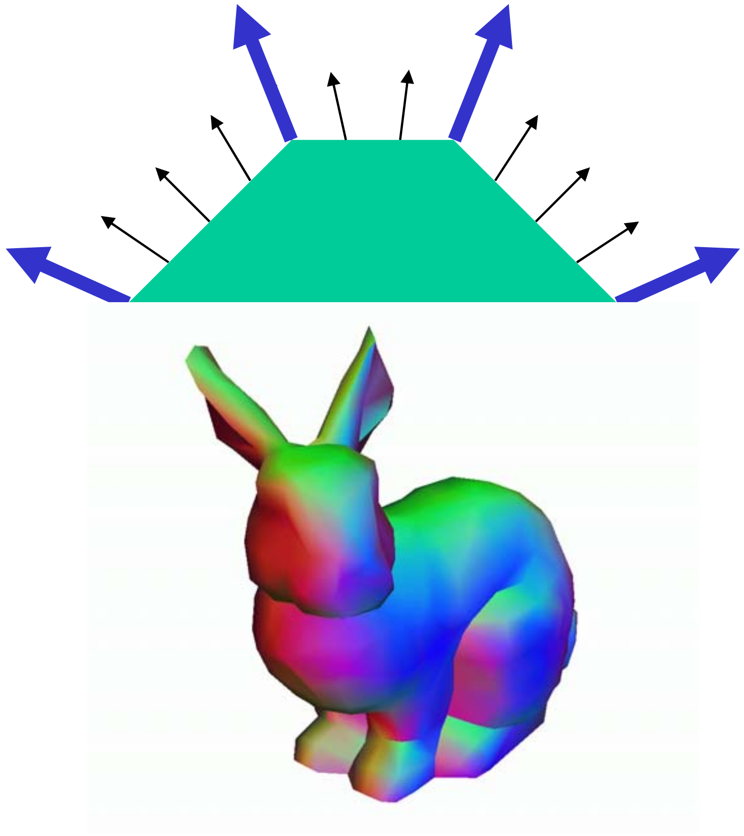


Graphics Lecture 8: Slide 40

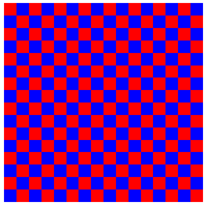
Remember Normal Averaging for Shading?

- Instead of using the normal of the triangle, interpolate an averaged normal at each vertex across the face

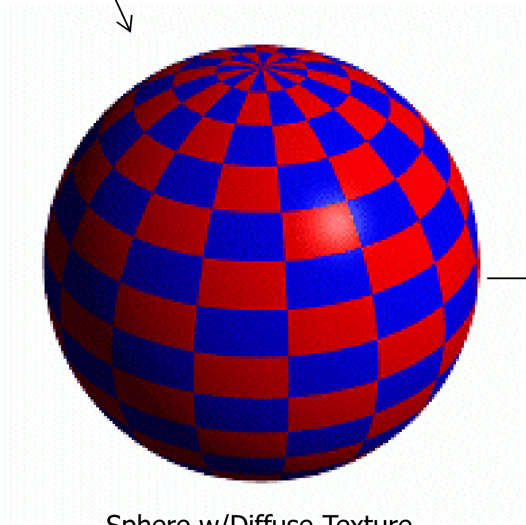




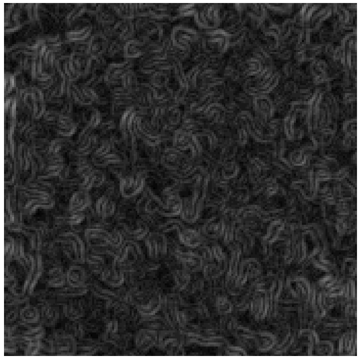
Bump Mapping



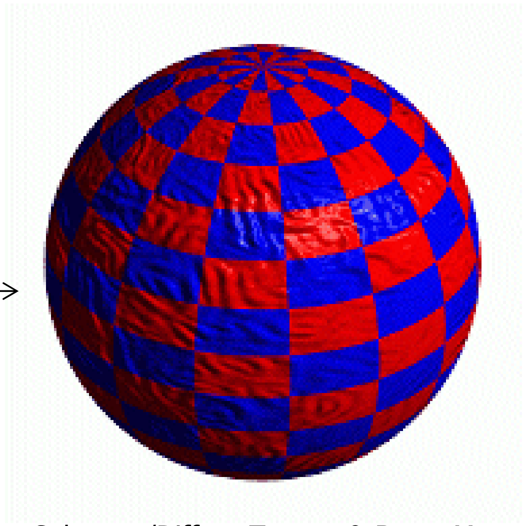
- Textures can be used to alter the surface normal of an object.
- Does not change actual shape of the surface – we only shade it as if it were a different shape!



Sphere w/Diffuse Texture



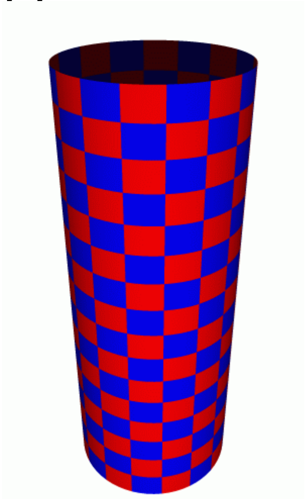
Swirly Bump Map

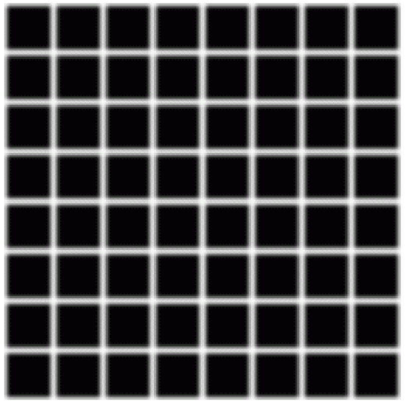


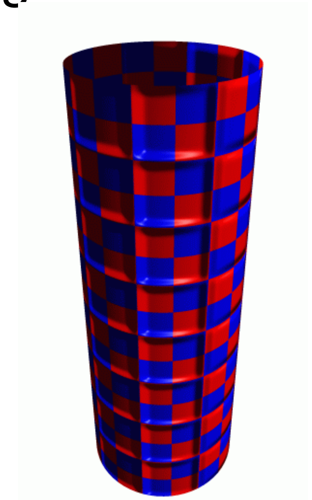
Sphere w/Diffuse Texture & Bump Map

Bump Mapping

- The texture map is treated as a single-valued height function.
- The partial derivatives of the texture tell us how to alter the true surface normal at each point to make the object appear as if it were deformed by the height function.


Cylinder w/Diffuse Texture Map


Bump Map


Cylinder w/Texture Map & Bump Map

Another Bump Map Example

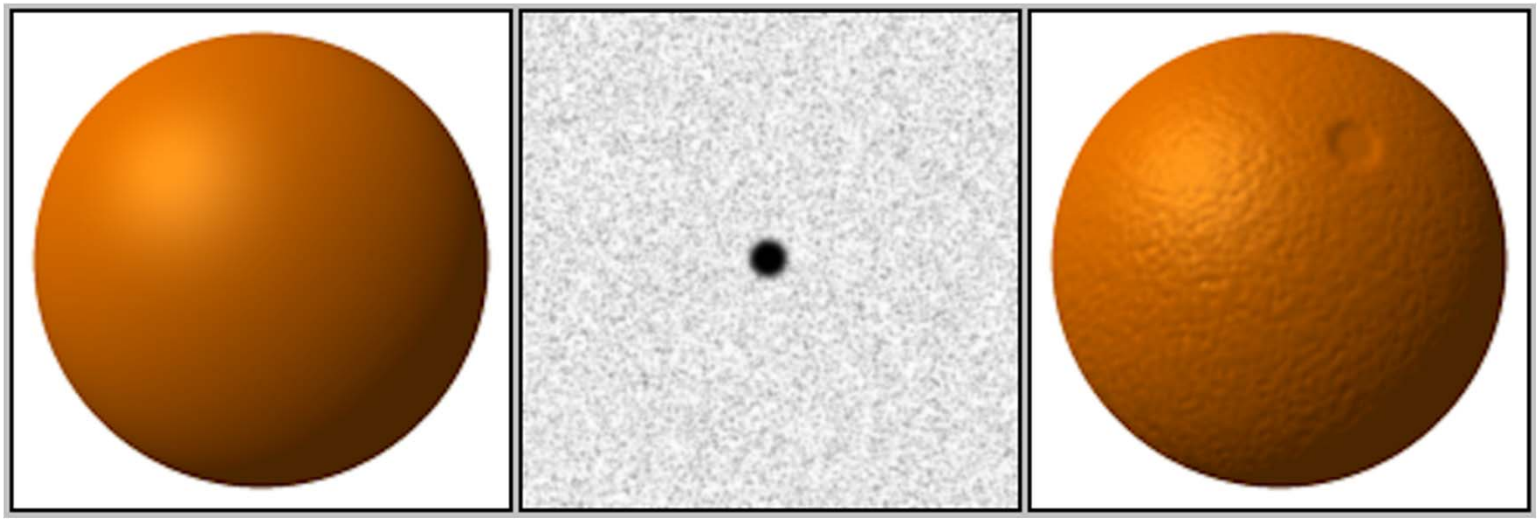
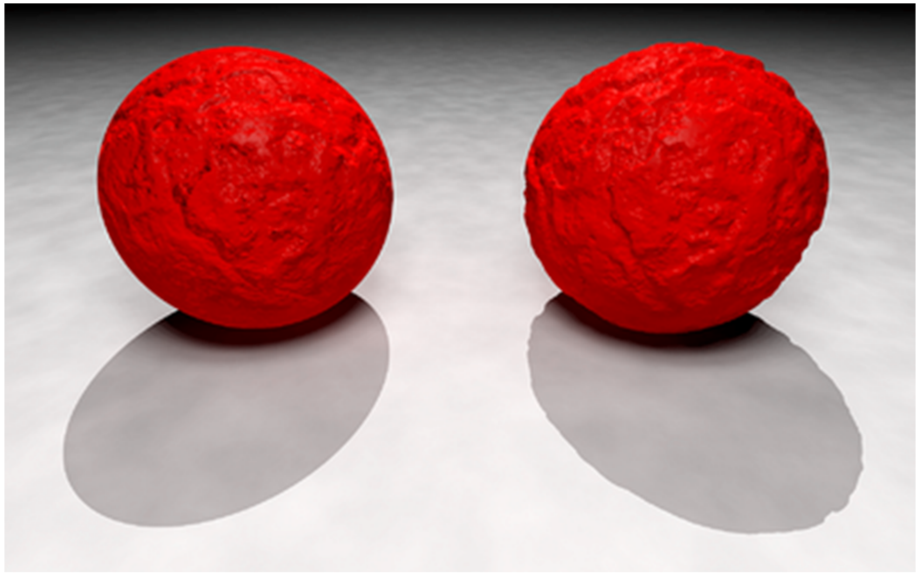


Image source: Wikipedia, 2016

What's Missing?

- What does a texture- & bump-mapped object look like as you move the viewpoint?

- What does the silhouette of a bump-mapped object look like?

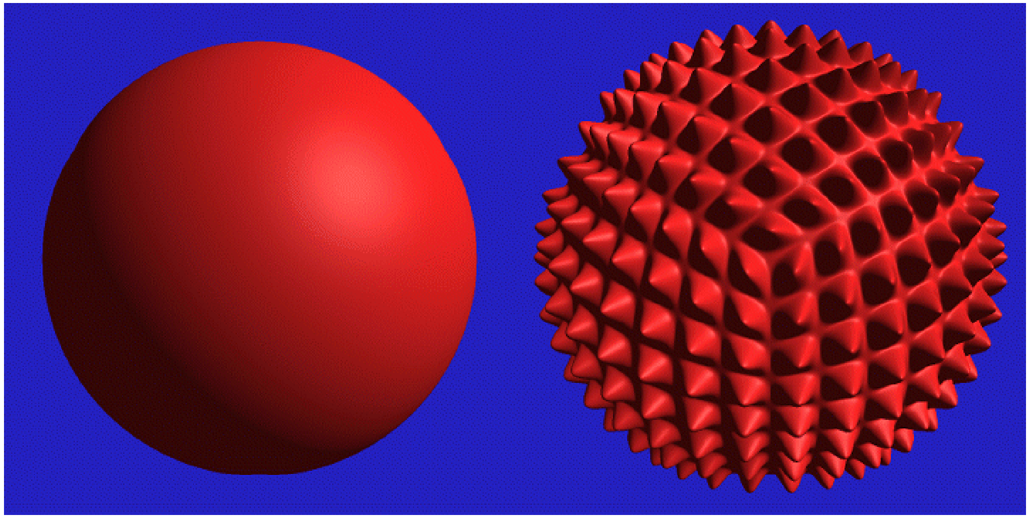


https://threejs.org/examples/webgl_materials_bumpmap.html

Image source: Wikipedia, 2016

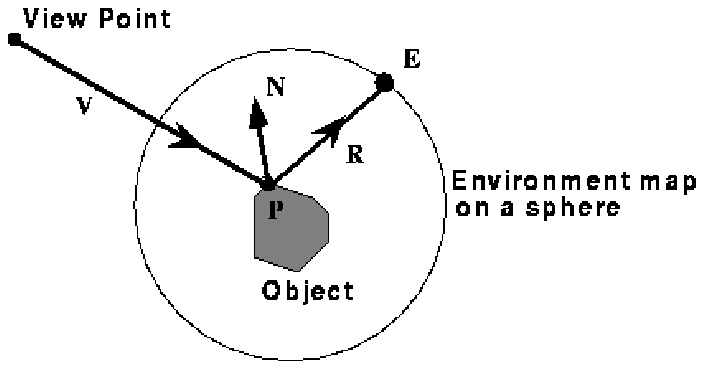
Displacement Mapping

- Use the texture map to actually move the surface point.
  - How is this different than bump mapping?
- The geometry must be displaced before visibility is determined.

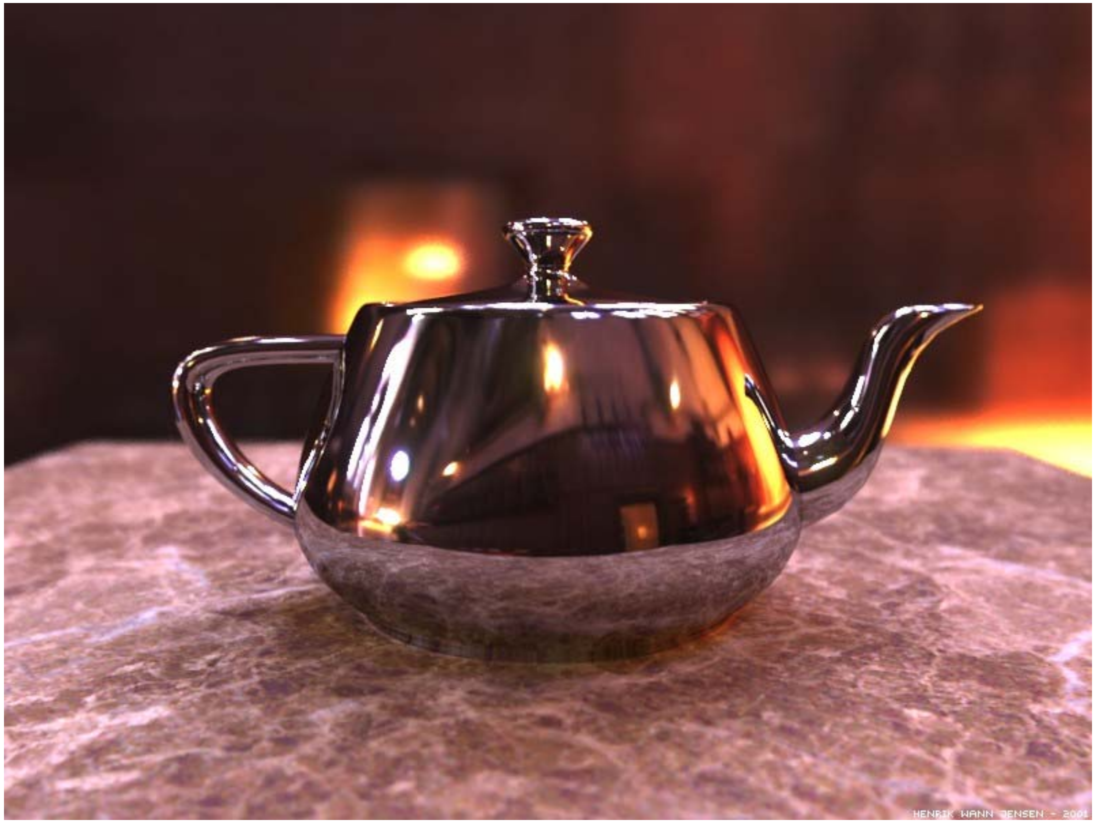


Environment Maps

- We can simulate reflections by using the direction of the reflected ray to index a spherical texture map at "infinity".
- Assumes that all reflected rays begin from the same point.



Environment Mapping Example



https://threejs.org/examples/webgl_materials_cubemap.html

Environment Mapping Example

Graphics Lecture 8: Slide 49

Interactive examples

- https://threejs.org/examples/#webgl_materials_bumpmap
- https://threejs.org/examples/#webgl_materials_displacementmap
- https://threejs.org/examples/webgl_materials_cubemap_dynamic2.html
- https://www.youtube.com/watch?v=K5n3p97-tuQ

Interactive Computer Graphics: Lecture 9

Rasterization, Visibility & Anti-aliasing

Some slides adopted from
F. Durand and B. Cutler, MIT

The Graphics Pipeline

Modelling Transformations

Illumination (Shading)

Viewing Transformation (Perspective / Orthographic)

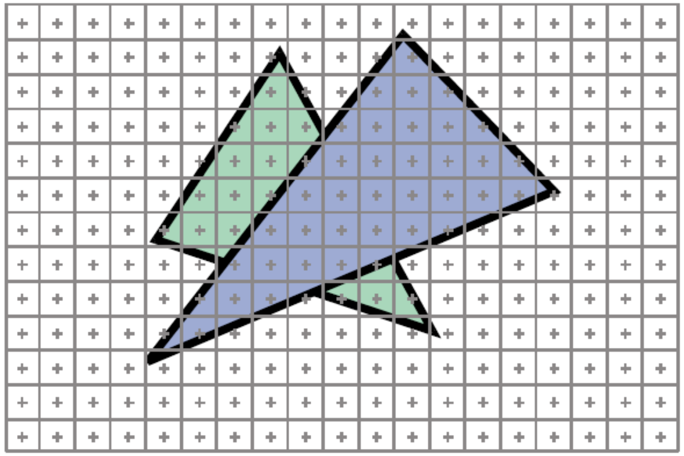
Clipping

Projection (to Screen Space)

Scan Conversion (Rasterization)

Visibility / Display

- Rasterizes objects into pixels
- Interpolate values inside objects (color, depth, etc.)



Graphics Lecture 9: Slide 2

The Graphics Pipeline

Modelling Transformations

Illumination (Shading)

Viewing Transformation (Perspective / Orthographic)

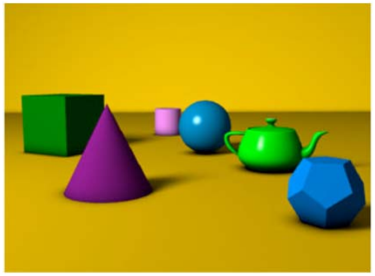
Clipping

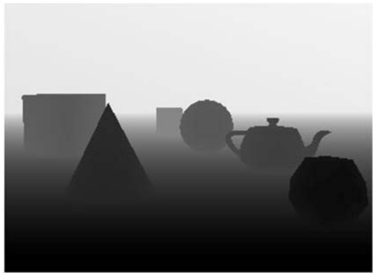
Projection (to Screen Space)

Scan Conversion (Rasterization)

Visibility / Display

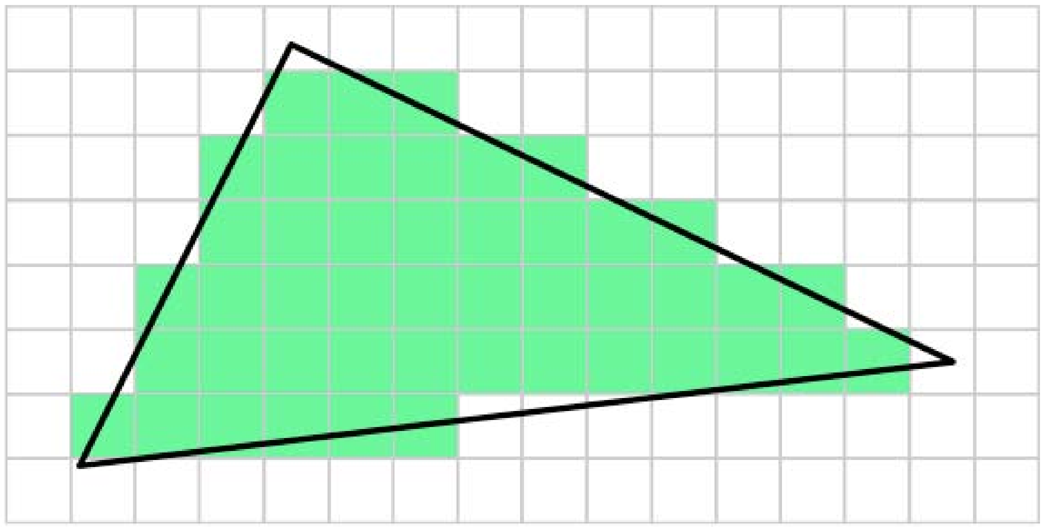
- Handles occlusions
- Determines which objects are closest and therefore visible





Rasterization

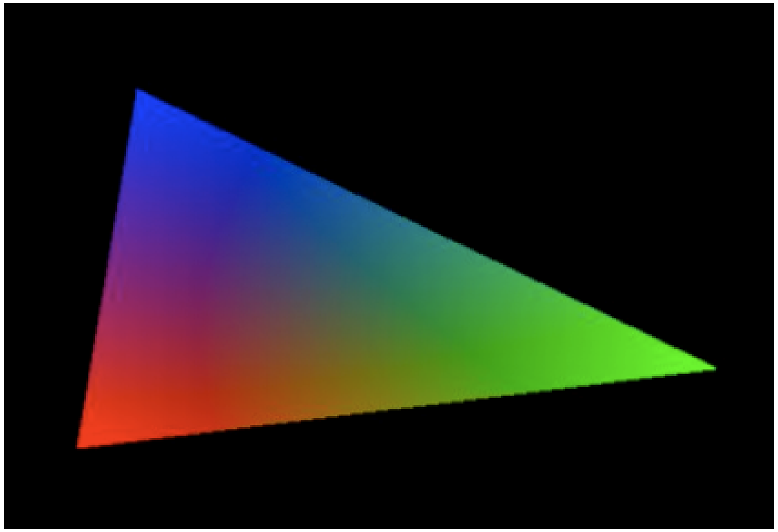
- Determine which pixels are drawn into the framebuffer
- Interpolate parameters (colors, texture coordinates, etc.)



Graphics Lecture 9: Slide 4

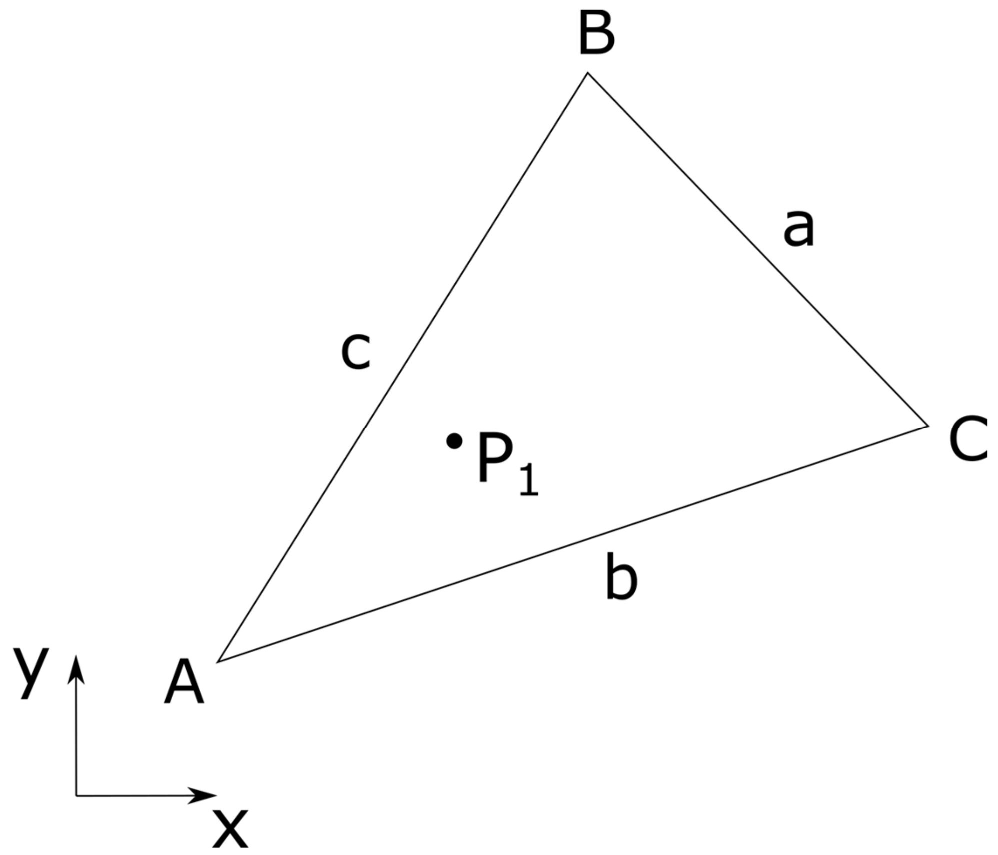
Rasterization

- What does interpolation mean?
- Examples: Colors, normals, shading, texture coordinates



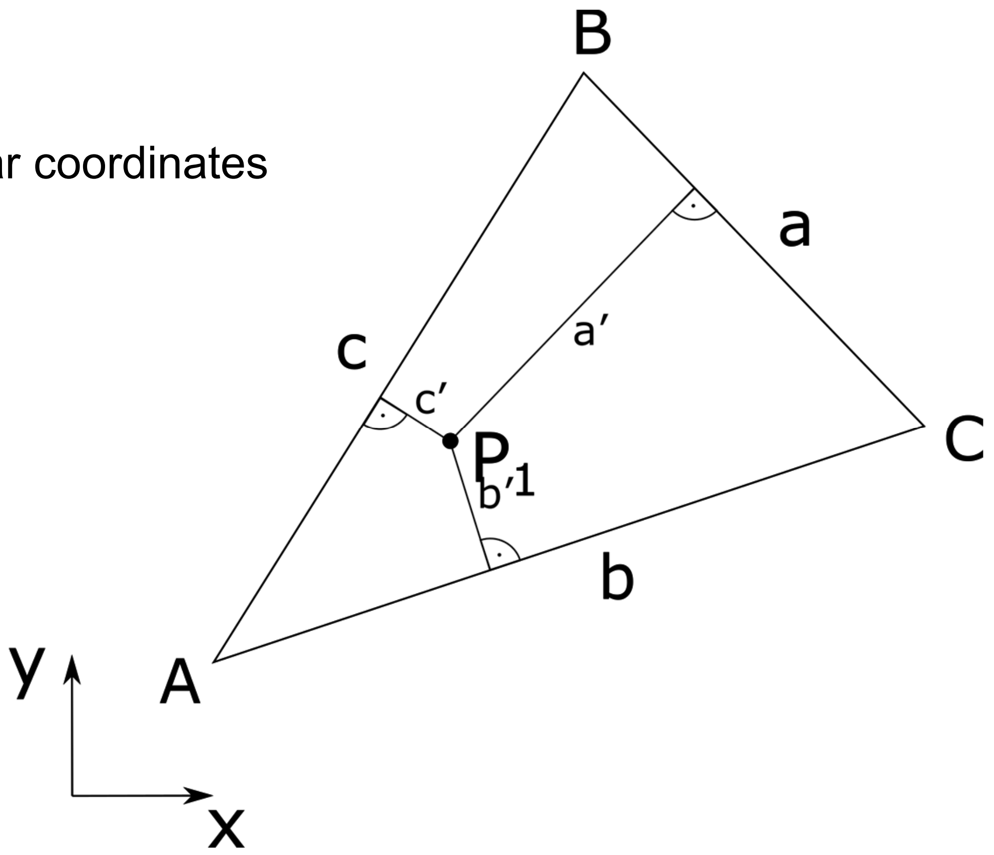
Graphics Lecture 9: Slide 5

Coordinate intuition



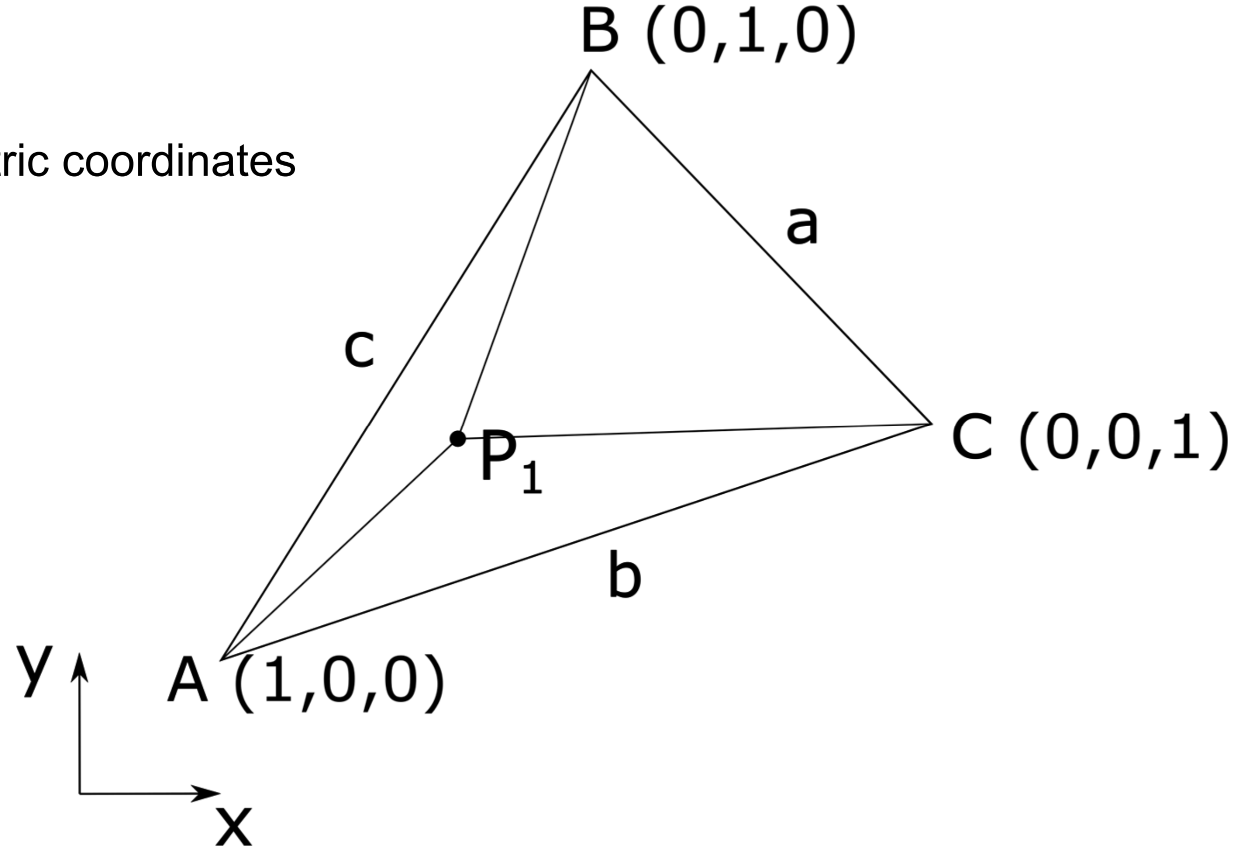
Coordinate intuition

Trilinear coordinates



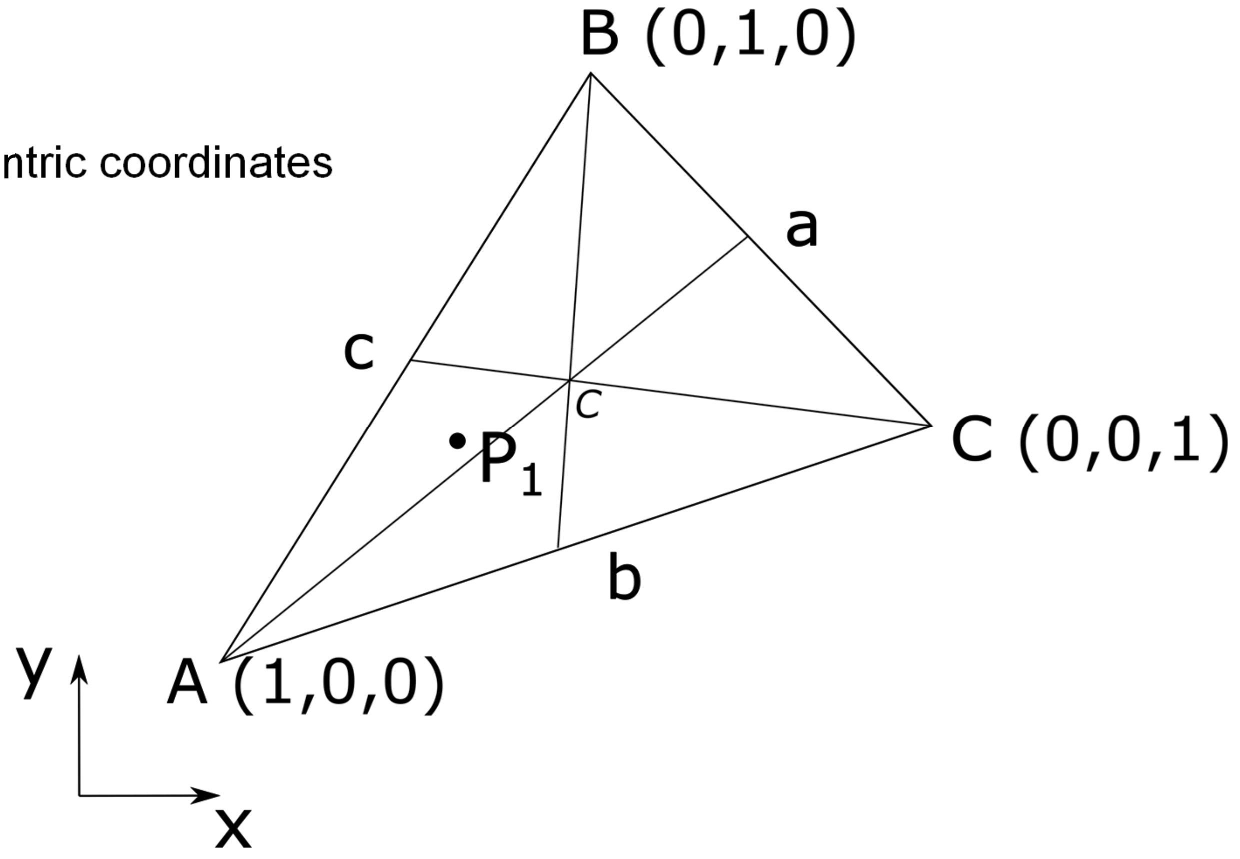
Coordinate intuition

barycentric coordinates



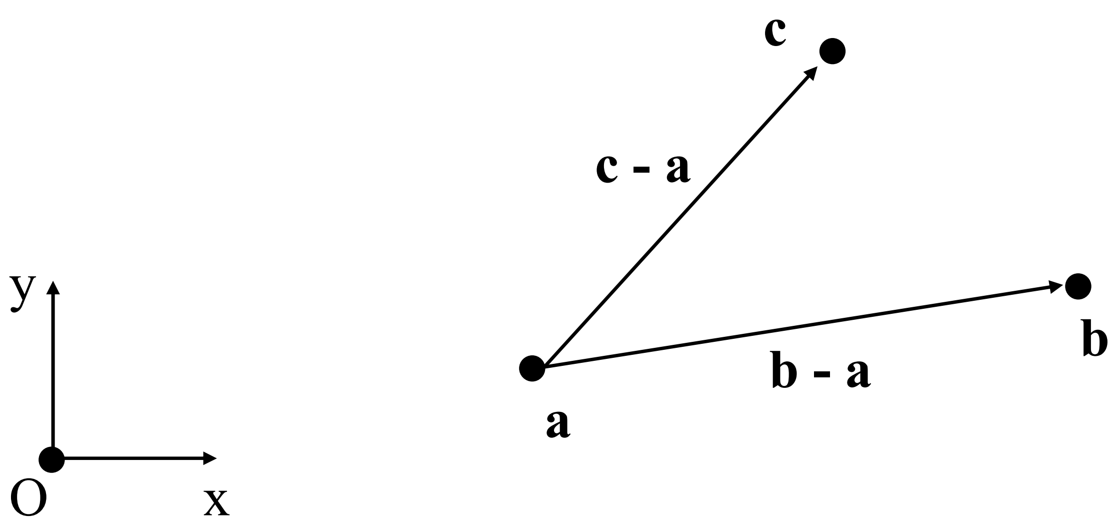
Coordinate intuition

barycentric coordinates



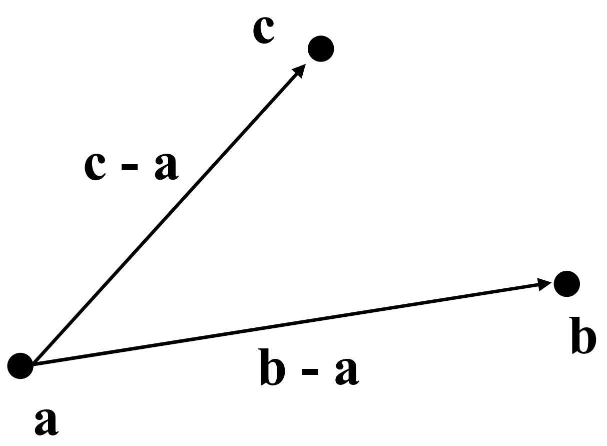
A triangle in terms of vectors

- We can use vertices **a**, **b** and **c** to specify the three points of a triangle
- We can also compute the edge vectors



Points and planes

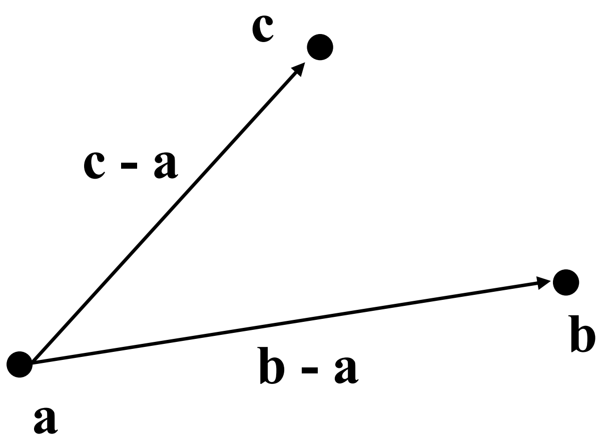
- The three non-collinear points determine a plane



- Example: The vertices **a**, **b** and **c** determine a plane
- The vectors **b - a** and **c - a** form a basis for this plane

Basis vectors

- This (non-orthogonal) basis can be used to specify the location of any point $\mathbf{p}$ in the plane



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

Barycentric coordinates

- We can reorder the terms of the equation:

$$\begin{aligned}\mathbf{p} &= \mathbf{a} + \beta(\mathbf{b} - \mathbf{a}) + \gamma(\mathbf{c} - \mathbf{a}) \\ &= (1 - \beta - \gamma)\mathbf{a} + \beta\mathbf{b} + \gamma\mathbf{c} \\ &= \alpha\mathbf{a} + \beta\mathbf{b} + \gamma\mathbf{c}\end{aligned}$$

- In other words:

$$\mathbf{p}(\alpha, \beta, \gamma) = \alpha\mathbf{a} + \beta\mathbf{b} + \gamma\mathbf{c}$$

- α, β, γ and called barycentric coordinates

Barycentric coordinates

- **Homogenous barycentric coordinates:**
  - normalised so that $\alpha + \beta + \gamma =$ area of triangle

- **Areal coordinates or absolute barycentric coordinates** : barycentric coordinates *normalized by the area of the original triangle* $\alpha + \beta + \gamma = 1$

Barycentric coordinates

- Barycentric coordinates describe a point $\mathbf{p}$ as an affine combination of the triangle vertices

$$\mathbf{p}(\alpha,\beta,\gamma) = \alpha\mathbf{a} + \beta\mathbf{b} + \gamma\mathbf{c} \qquad \alpha + \beta + \gamma = 1$$

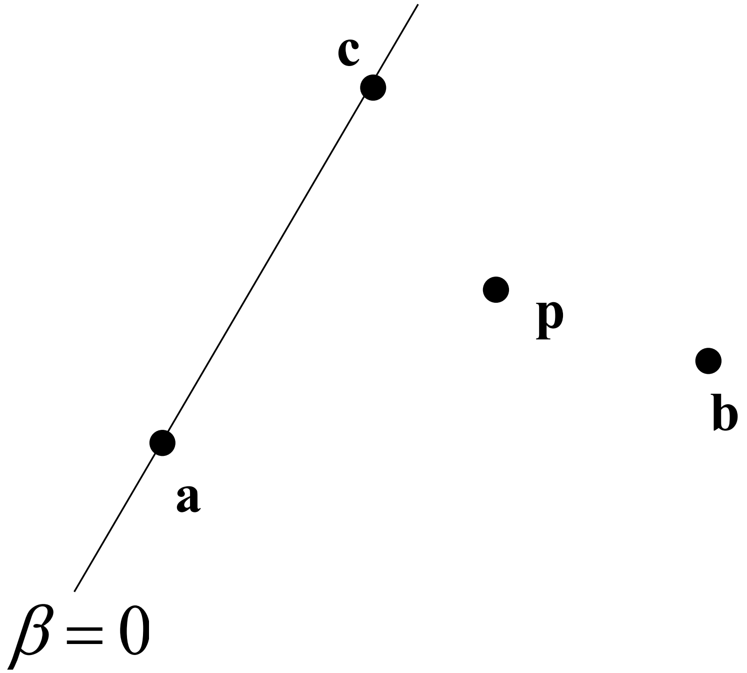
- For any point $\mathbf{p}$ inside the triangle $(\mathbf{a}, \mathbf{b}, \mathbf{c})$:

$$0 < \alpha < 1$$
$$0 < \beta < 1$$
$$0 < \gamma < 1$$

- Point on an edge: one coefficient is 0
- Vertex: two coefficients are 0, remaining one is 1

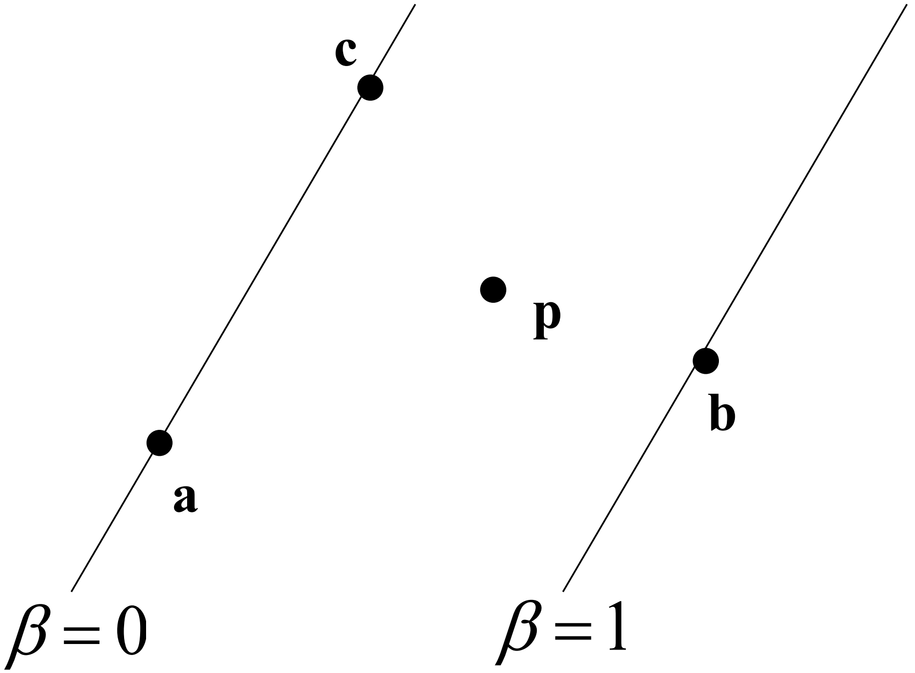
Barycentric coordinates and signed distances

- Let $\mathbf{p} = \alpha\mathbf{a}+\beta\mathbf{b}+\gamma\mathbf{c}$. Each coordinate (e.g. β) is the signed distance from $\mathbf{p}$ to the line through a triangle edge (e.g. $\mathbf{ac}$)



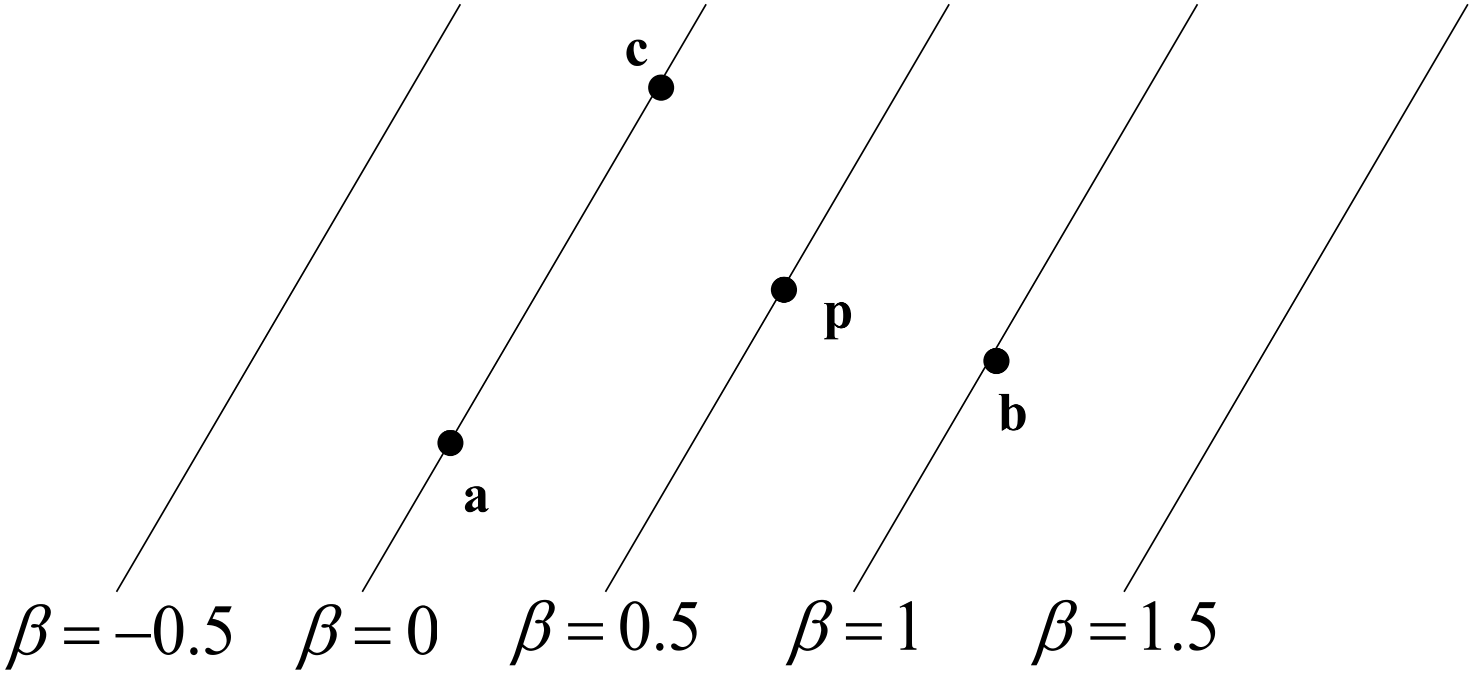
Barycentric coordinates and signed distances

- Let $\mathbf{p} = \alpha\mathbf{a}+\beta\mathbf{b}+\gamma\mathbf{c}$. Each coordinate (e.g. β) is the signed distance from $\mathbf{p}$ to the line through a triangle edge (e.g. $\mathbf{ac}$)



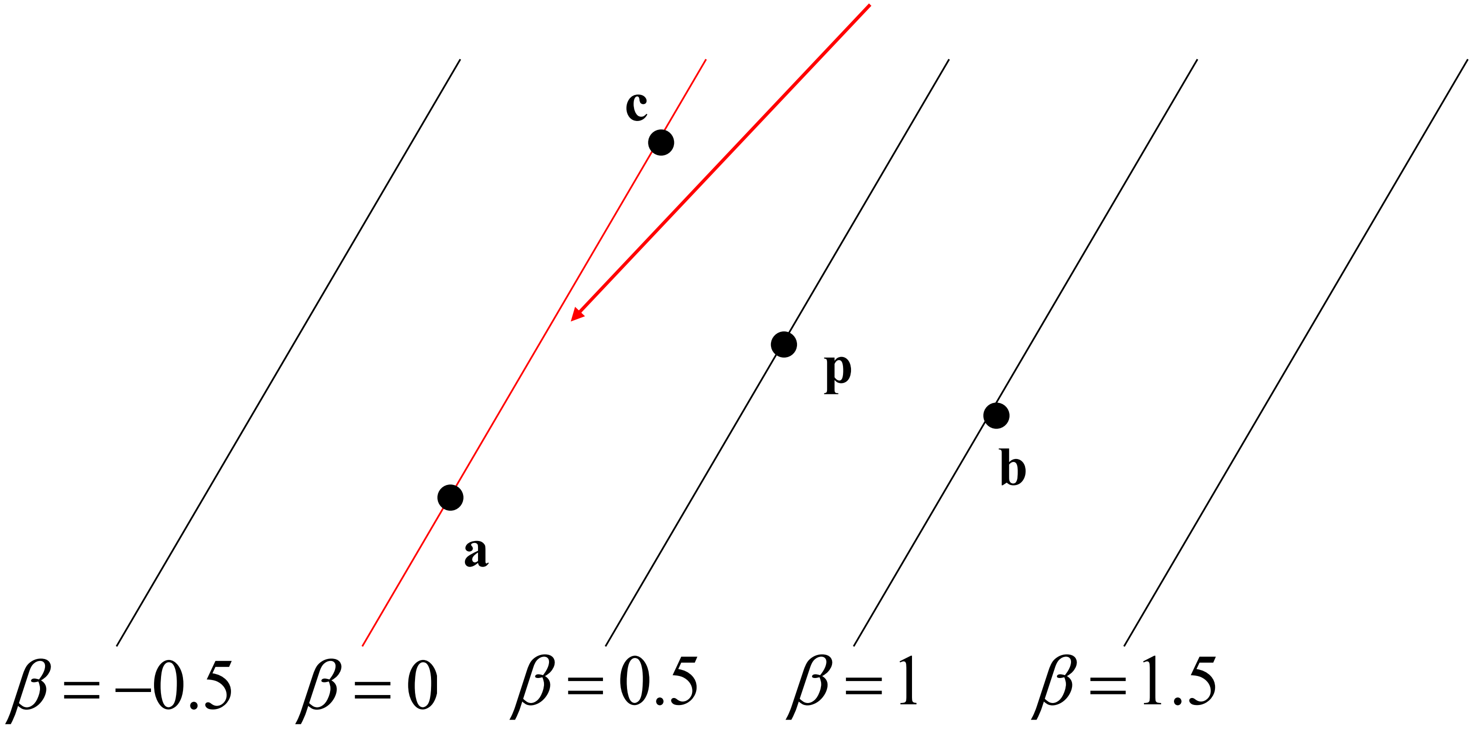
Barycentric coordinates and signed distances

- Let $\mathbf{p} = \alpha\mathbf{a}+\beta\mathbf{b}+\gamma\mathbf{c}$. Each coordinate (e.g. β) is the signed distance from $\mathbf{p}$ to the line through a triangle edge (e.g. $\mathbf{ac}$)



Barycentric coordinates and signed distances

- The signed distance can be computed by evaluating implicit line equations, e.g., $f_{ac}(x,y)$ of edge **ac**



Recall: Implicit equation for lines

- Implicit equation in 2D:

$$f(x,y)=0$$

  - Points with $f(x, y) = 0$ are on the line
  - Points with $f(x, y) \neq 0$ are not on the line
- General implicit form

$$Ax + By + C = 0$$

- Implicit line through two points (x_a, y_a) and (x_b, y_b)

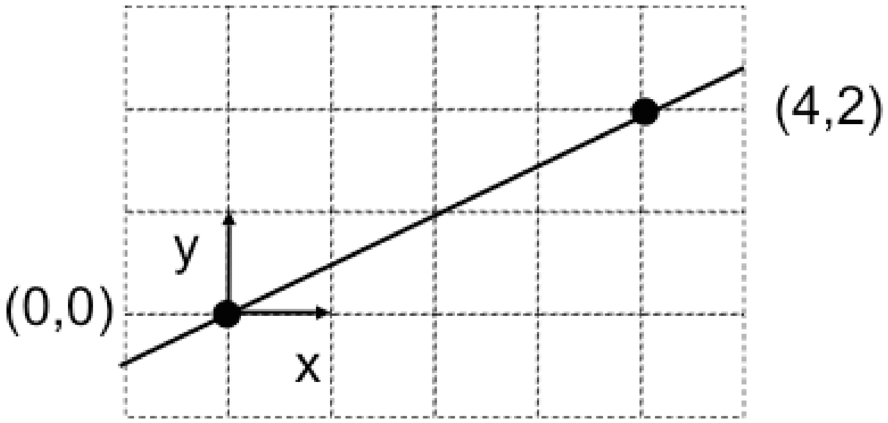
$$(y_a - y_b)x + (x_b - x_a)y + x_a y_b - x_b y_a = 0$$

Implicit equation for lines: Example

$A =$

$B =$

$C =$

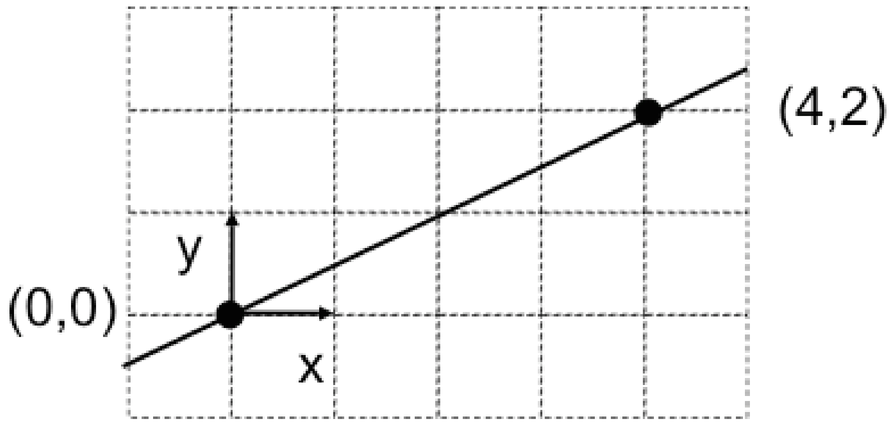


Implicit equation for lines: Example

Solution 1: $-2x + 4y = 0$

Solution 2: $2x - 4y = 0$

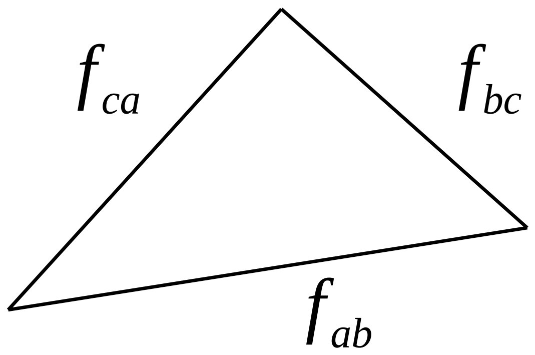
$$kf(x,y) = 0 \text{ for any } k$$



Edge equations

- Given a triangle with vertices (x_a,y_a), (x_b,y_b), and (x_c,y_c).
- The line equations of the edges of the triangle are:

$$f_{ab}(x,y) = (y_a - y_b)x + (x_b - x_a)y + x_a y_b - x_b y_a$$
$$f_{bc}(x,y) = (y_b - y_c)x + (x_c - x_b)y + x_b y_c - x_c y_b$$
$$f_{ca}(x,y) = (y_c - y_a)x + (x_a - x_c)y + x_c y_a - x_a y_c$$



Barycentric Coordinates

- Remember that: $f(x,y)=0 \Leftrightarrow kf(x,y)=0$
- A barycentric coordinate (e.g. β) is a signed distance from a line (e.g. the line that goes through **ac**)
- For a given point **p**, we would like to compute its barycentric coordinate β using an implicit edge equation.
- We need to choose k such that

$$kf_{ac}(x,y)=\beta$$

Barycentric Coordinates

- We would like to choose k such that: $kf_{ac}(x,y) = \beta$
- We know that β = 1 at point **b**:

$$kf_{ac}(x,y) = 1 \Leftrightarrow k = \frac{1}{f_{ac}(x_b, y_b)}$$

- The barycentric coordinate β for point **p** is:

$$\beta = \frac{f_{ac}(x,y)}{f_{ac}(x_b, y_b)}$$

Barycentric Coordinates

- In general, the barycentric area coordinates for point $\mathbf{p}$ are:

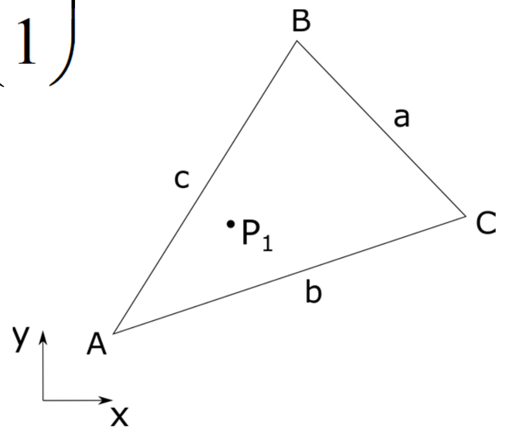
$$\alpha = \frac{f_{bc}(x,y)}{f_{bc}(x_a,y_a)} \qquad \beta = \frac{f_{ac}(x,y)}{f_{ac}(x_b,y_b)} \qquad \gamma = 1 - \alpha - \beta$$

- Given a point $\mathbf{p}$ with Cartesian coordinates (x, y), we can compute its barycentric coordinates (α, β, γ) as above.

Barycentric Coordinates

- In general, the barycentric area coordinates for point **p** are the solution of the linear system of equations:

$$\begin{pmatrix} x_a & x_b & x_c \\ y_a & y_b & y_c \\ 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \\ \gamma \end{pmatrix} = \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$



Barycentric Coordinates

- Can be easily converted to trilinear coordinates

P_t (t_1, t_2, t_3) in trilinear coordinates has barycentric coordinates of **(t_1 a, t_2 b, t_3 c)**

where **a, b, c,** are the side lengths of the triangle.

P_b (α, ß, γ) in barycentric coordinates has trilinear coordinates **(α/a, ß/b, γ/c)**

Triangle Rasterization

- Many different ways to generate fragments for a triangle
- Checking (α, β, γ) is one method, e.g.

$$(0< \alpha <1 \ \&\& \ 0< \beta <1 \ \&\& \ 0 < \gamma <1)$$

- In practice, the graphics hardware uses optimized methods:
  - fixed point precision (not floating-point)
  - incremental (use results from previous pixel)

Triangle Rasterization

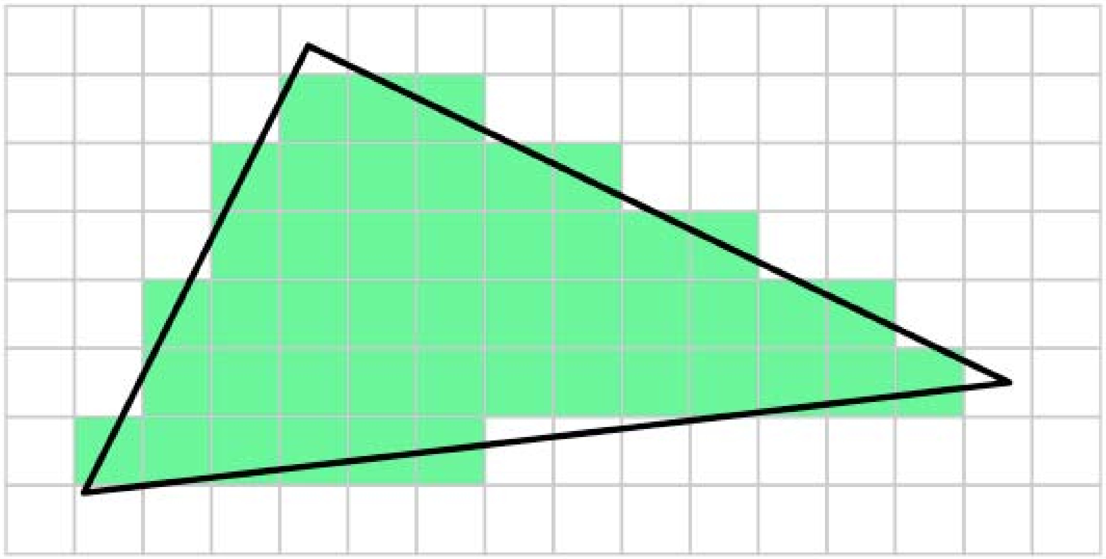
- We can use barycentric coordinates to rasterize and color triangles

```
for all x do
    for all y do
       compute (alpha, beta, gamma) for (x,y)
       if (0 < alpha < 1 and
           0 < beta  < 1 and
           0 < gamma < 1 ) then
           c = alpha c0 + beta c1 + gamma c2
           drawpixel(x,y) with color c
```

- The color c varies smoothly within the triangle

Visibility: One triangle

- With one triangle, things are simple
- Pixels never overlap!



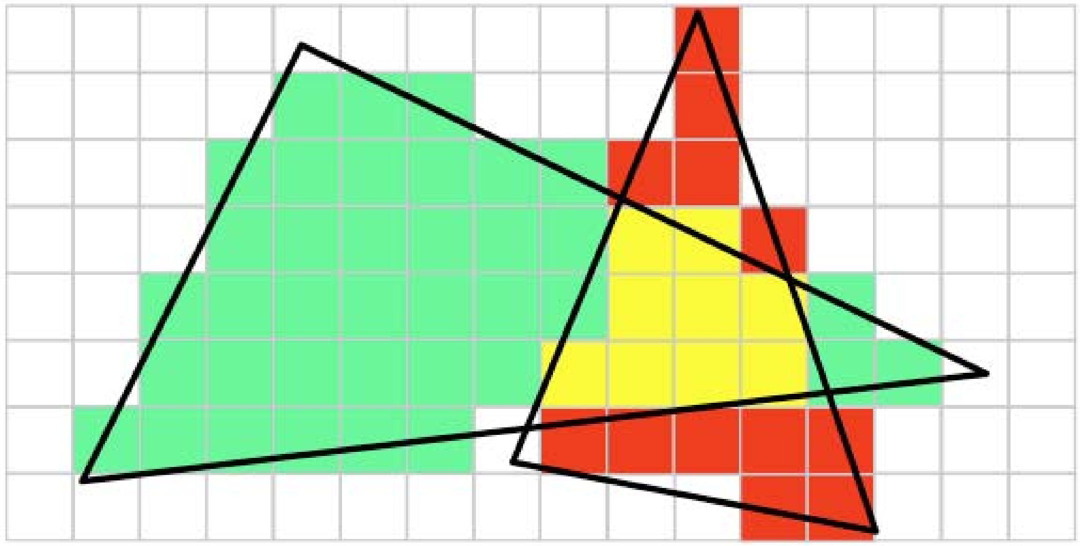
Graphics Lecture 9: Slide 31

Hidden Surface Removal

- Idea: keep track of visible surfaces
- Typically, we see only the front-most surface
- Exception: transparency

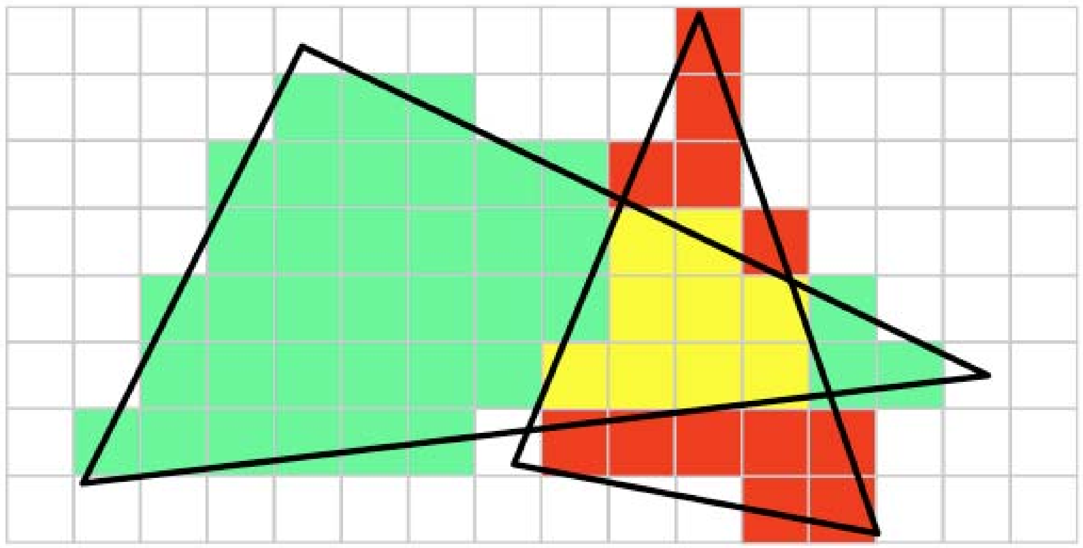
Visibility: Two triangles

- Things get more complicated with multiple triangles
- Fragments might overlap in screen space!



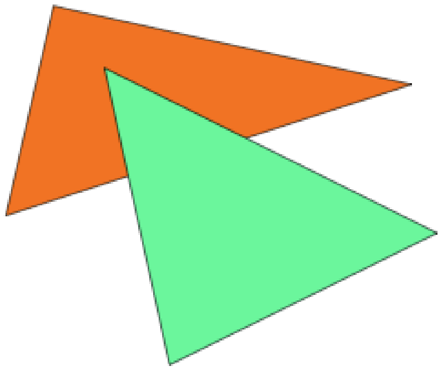
Visibility: Pixels vs Fragments

- Each pixel has a unique framebuffer (image) location
- But multiple fragments may end up at same address

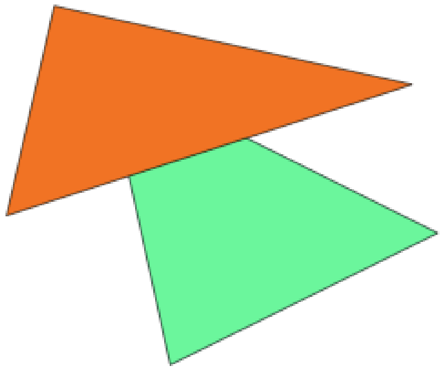


Visibility: Which triangle should be drawn first?

- Two possible cases:



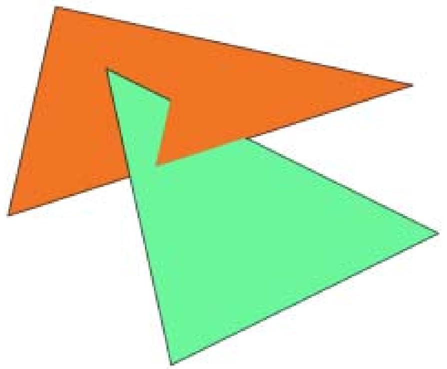
green triangle on top

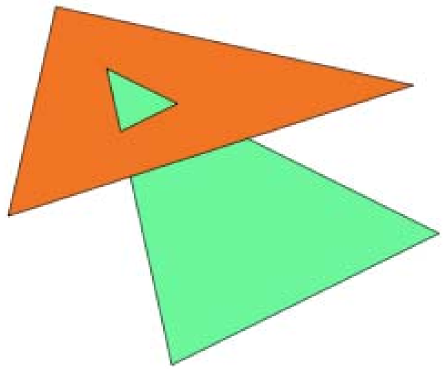


orange triangle on top

Visibility: Which triangle should be drawn first?

- Many other cases possible!

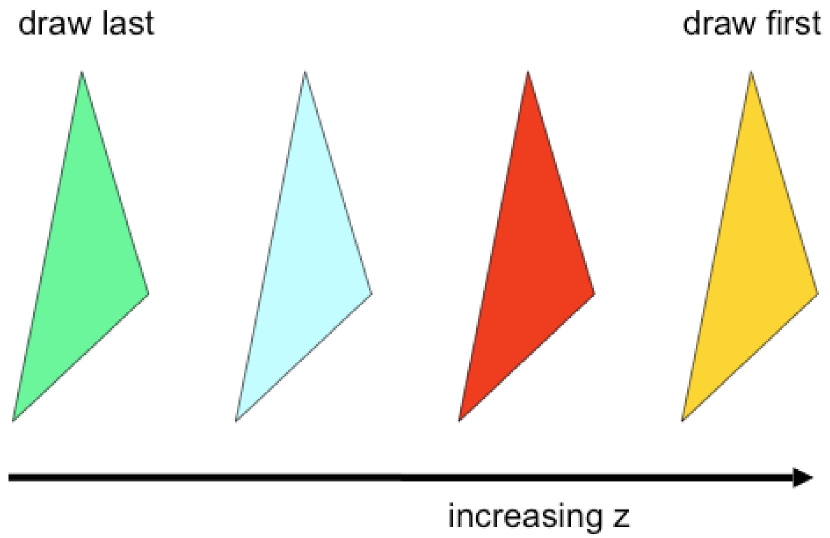

intersection #1


intersection #2

Visibility: Painter's Algorithm

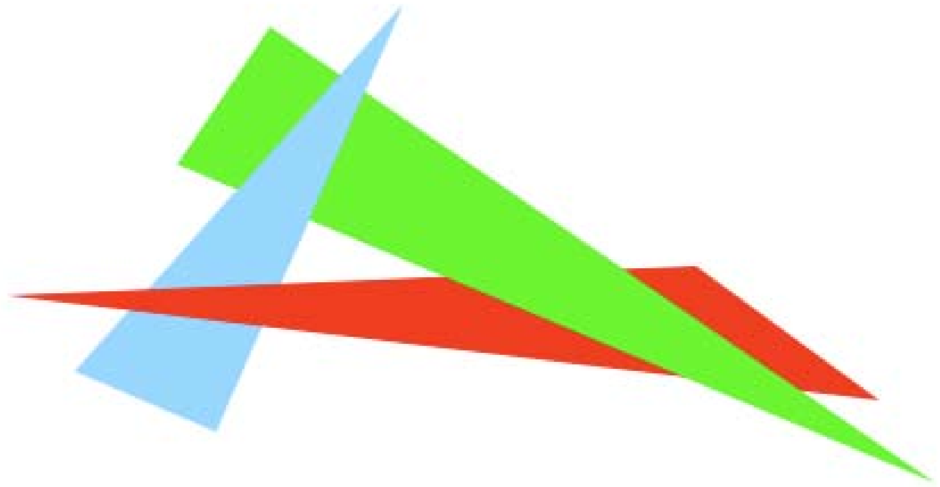
- Sort triangles (using z values in eye space)
- Draw triangles from back to front

Viewer



Visibility: Painter's Algorithm - Problems

- Correctness issues:
  - Intersections
  - Cycles
  - Solve by splitting triangles, but ugly and expensive
- Efficiency (sorting)

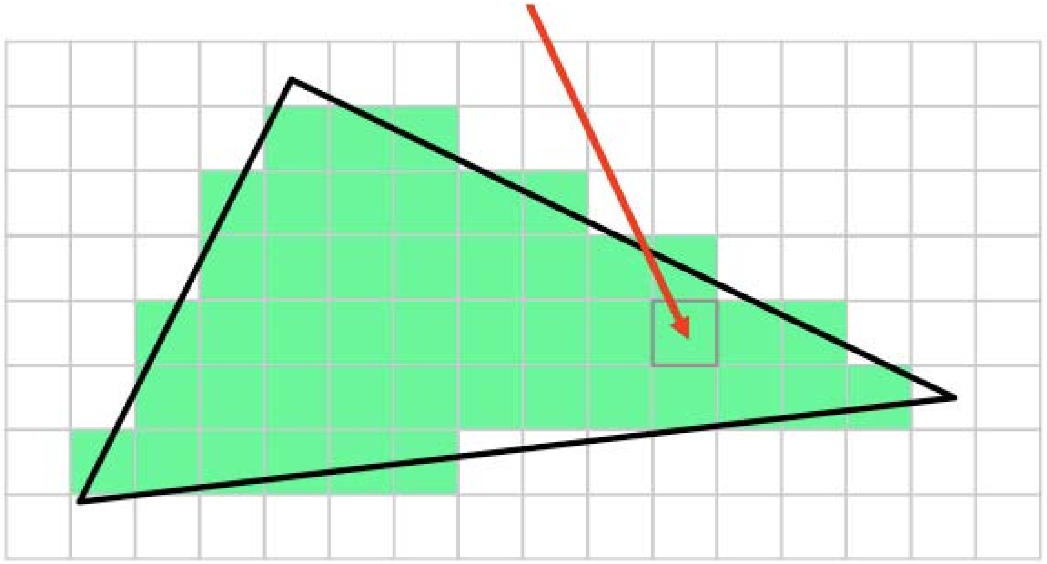


The Depth Buffer (Z-Buffer)

- Perform hidden surface removal per-fragment
- Idea:
  - Each fragment gets a z value in screen space
  - Keep only the fragment with the smallest z value

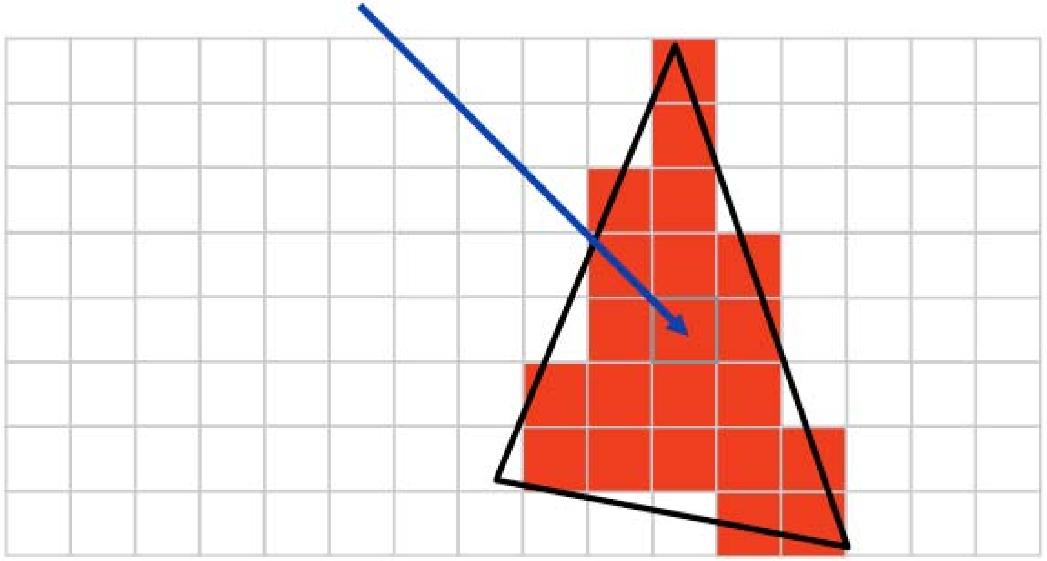
The Depth Buffer (Z-Buffer)

- Example:
  - fragment from green triangle has z value of 0.7



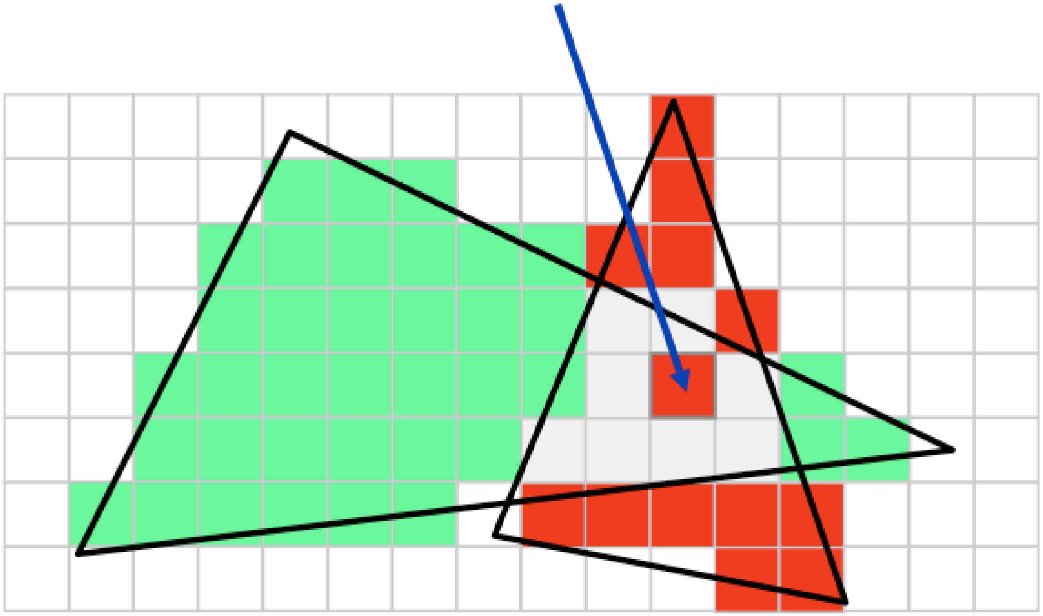
The Depth Buffer (Z-Buffer)

- Example:
  - fragment from red triangle has z value of 0.3



The Depth Buffer (Z-Buffer)

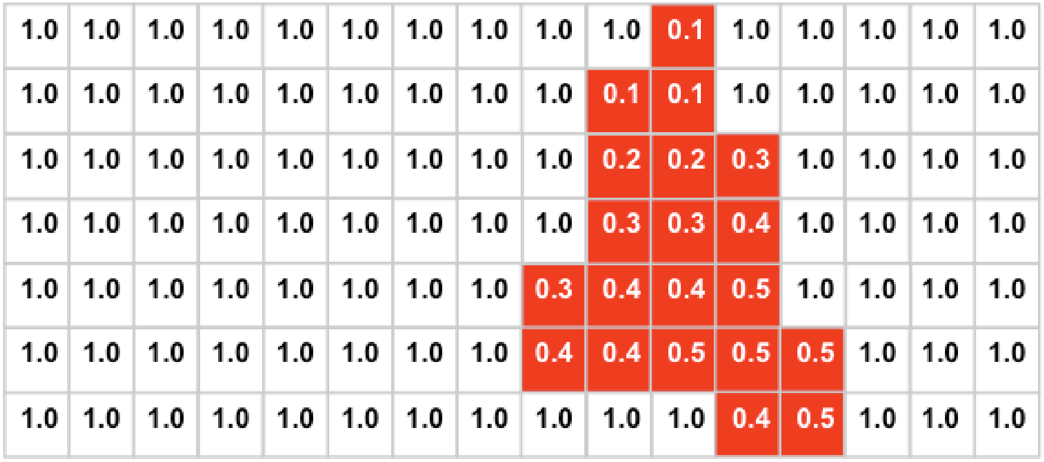
- Since 0.3 < 0.7, the red fragment wins



Graphics Lecture 9: Slide 42

The Depth Buffer (Z-Buffer)

- Many fragments might map to the same pixel location
- How to track their z-values?
- Solution: z-buffer (2D buffer, same size as image)



1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.1	1.0	1.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.1	0.1	1.0	1.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.2	0.2	0.3	1.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.3	0.3	0.4	1.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.3	0.4	0.4	0.5	1.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.4	0.4	0.5	0.5	0.5	1.0	1.0	1.0
1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.4	0.5	1.0	1.0	1.0

The Z-Buffer Algorithm

```
Let CB be color (frame) buffer, ZB be z-buffer
Initialize z-buffer contents to 1.0 (far away)
For each triangle T
   Rasterize T to generate fragments
   For each fragment F with screen position (x,y,z) and color value C
      If  (z < ZB[x,y])  then
           Update color:  CB[x,y] = C
           Update depth:  ZB[x,y] = z
```

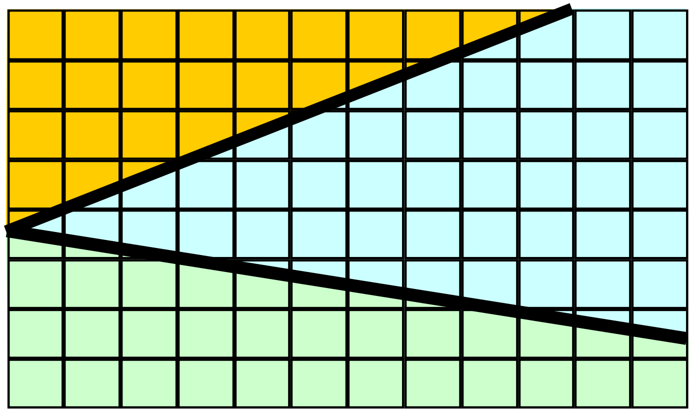
Z-buffer Algorithm Properties

- What makes this method nice?
  - simple (faciliates hardware implementation)
  - handles intersections
  - handles cycles
  - draw opaque polygons in any order

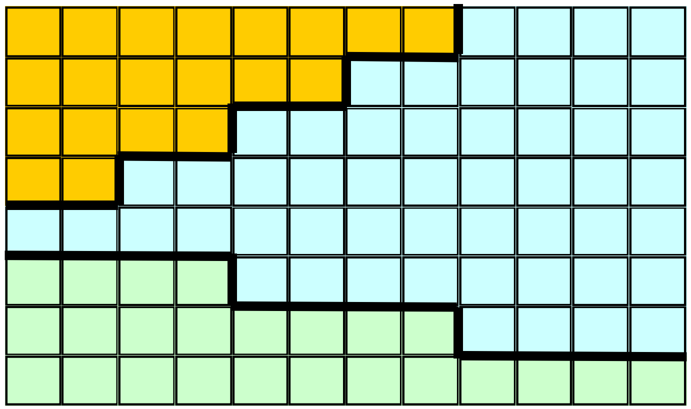
Alias Effects

- One major problem with rasterization is called alias effects, e.g straight lines or triangle boundaries look jagged
- These are caused by undersampling, and can cause unreal visual artefacts.
- It also occurs in texture mapping

Alias Effects at straight boundaries in raster images.

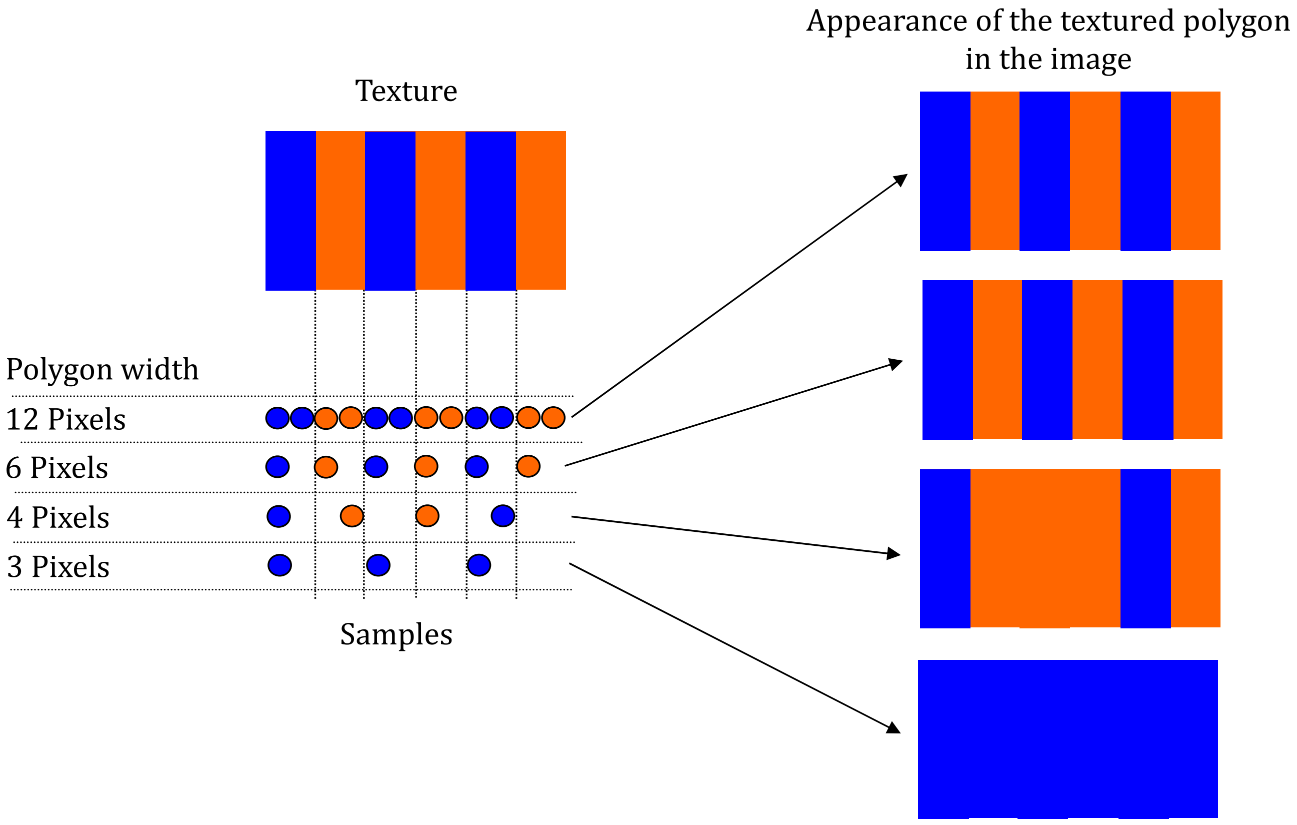


Desired Boundaries



Pixels Set

Graphics Lecture 9: Slide 47

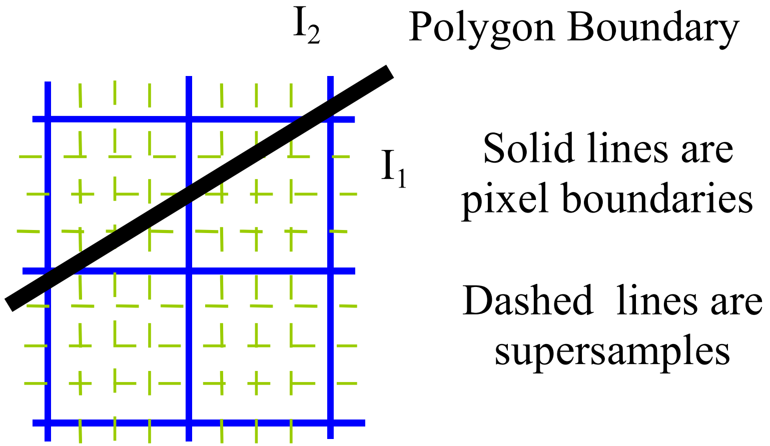

Appearance of the textured polygon in the image
Texture
Polygon width
12 Pixels
6 Pixels
4 Pixels
3 Pixels
Samples

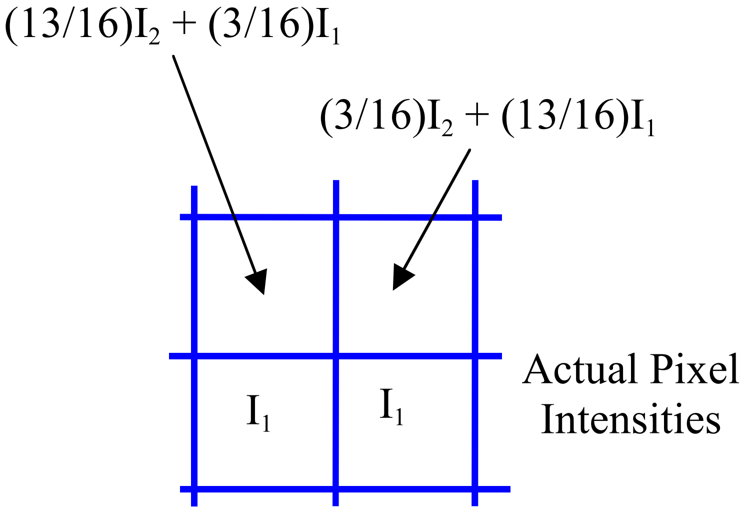
Anti-Aliasing

- The solution to aliasing problems is to apply a degree of blurring to the boundary such that the effect is reduced.
- The most successful technique is called **<u>Supersampling</u>**

Supersampling

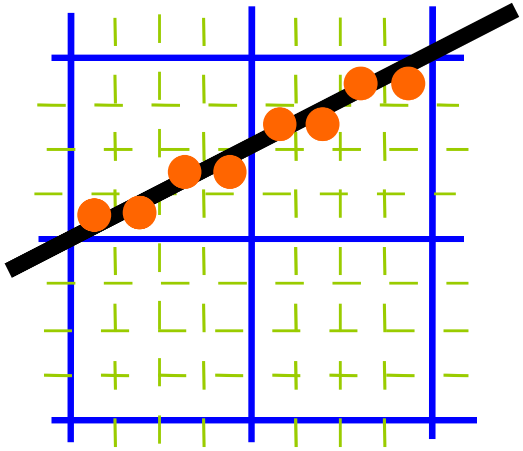
- The basic idea is to compute the picture at a higher resolution to that of the display area.
- Supersamples are averaged to find the pixel value.
- This has the effect of blurring boundaries, but leaving coherent areas of colour unchanged

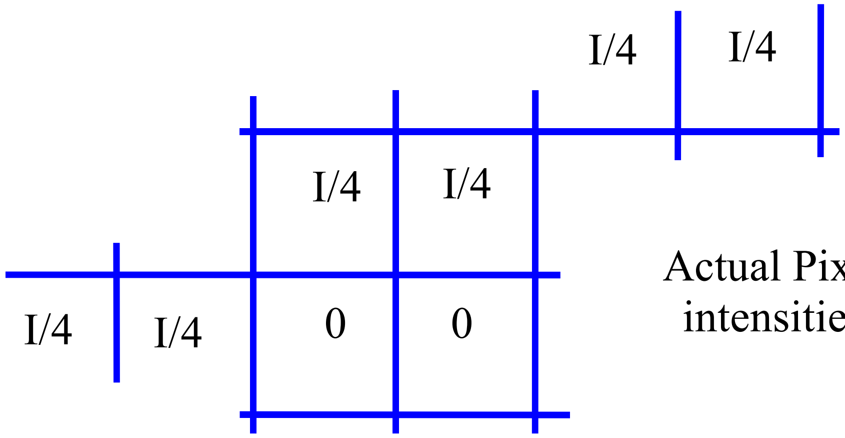

I_2
Polygon Boundary
I_1
Solid lines are pixel boundaries
Dashed lines are supersamples


$(13/16)I_2 + (3/16)I_1$
$(3/16)I_2 + (13/16)I_1$
I_1
I_1
Actual Pixel Intensities

Limitations of Supersampling

- Supersampling works well for scenes made up of filled polygons.
- However, it does require a lot of extra computation.
- It does not work for line drawings.

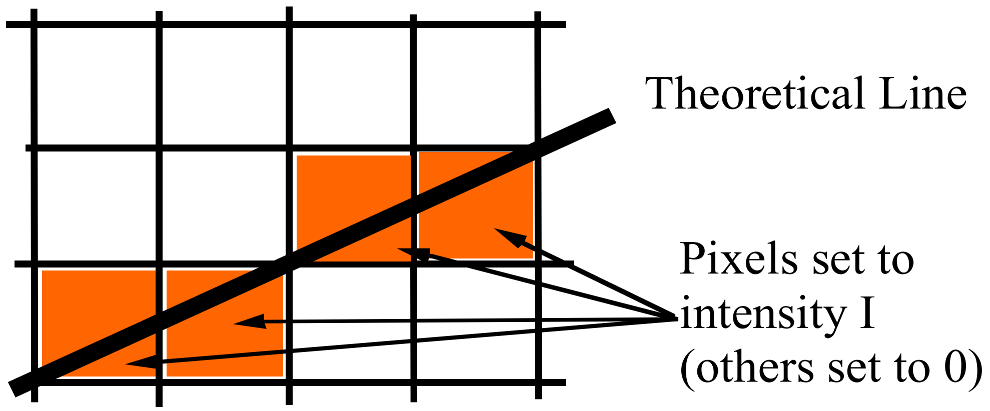



I/4
I/4
I/4
I/4
I/4
I/4
0
0
Actual Pixel intensities

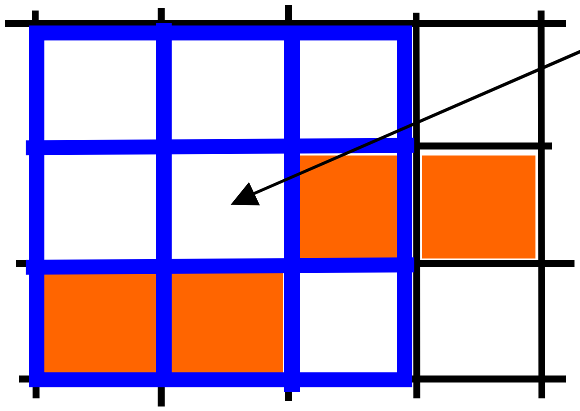
Convolution filtering

- The more common (and much faster) way of dealing with alias effects is to use a ‘filter’ to blur the image.
- This essentially takes an average over a small region around each pixel

For example consider the image of a line



Treat each pixel of the image



Consider one pixel.

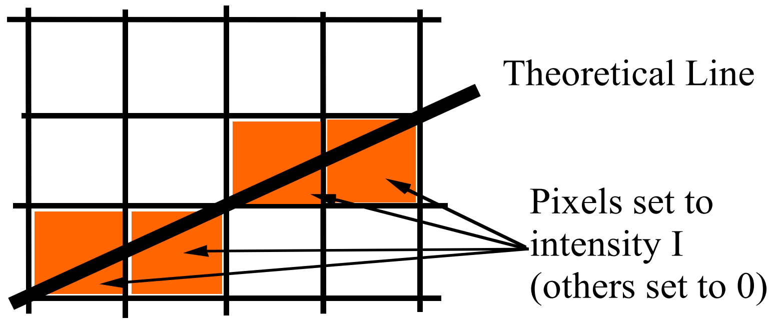
We replace the pixel by a local average, one possibility would be 3*I/9

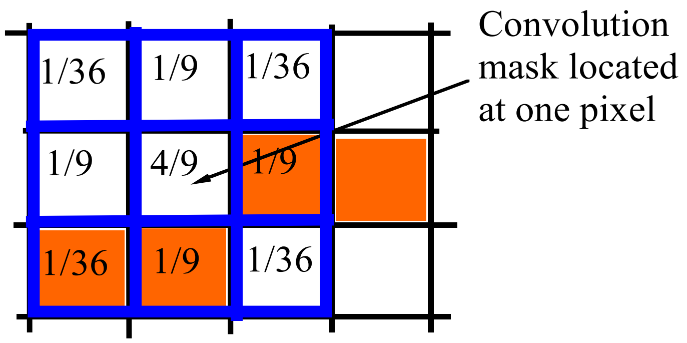
Weighted averages

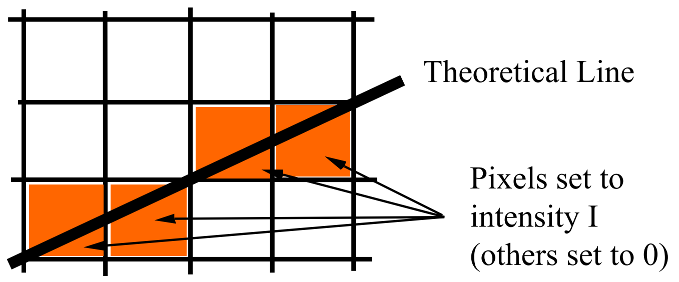
- Taking a straight local average has undesirable effects.
- Thus we normally use a weighted average.

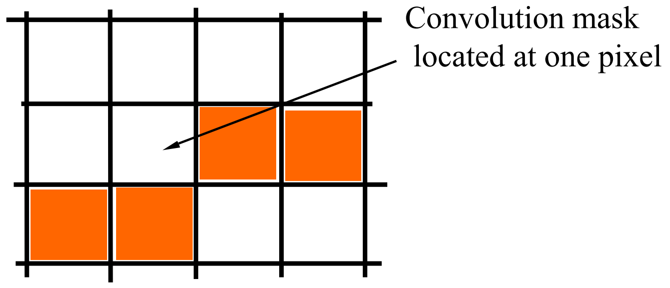
1/36 *

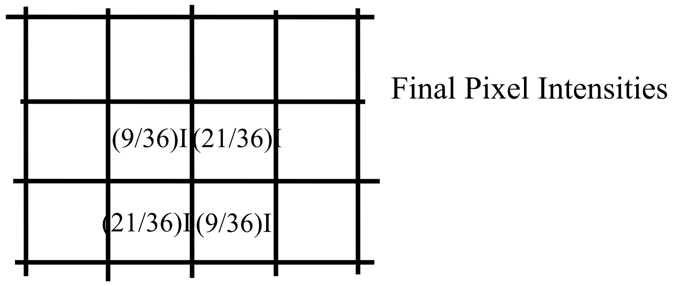
1	4	1
4	16	4
1	4	1


Theoretical Line
Pixels set to intensity I (others set to 0)


1/36
1/9
1/36
1/9
4/9
1/9
1/36
1/9
1/36
Convolution mask located at one pixel


Theoretical Line
Pixels set to intensity I (others set to 0)


Convolution mask located at one pixel


Final Pixel Intensities
(9/36)I
(21/36)I
(21/36)I
(9/36)I

Pros and Cons of Convolution filtering

- Advantages:
  - It is very fast and can be done in hardware
  - Generally applicable
- Disadvantages:
  - It does degrade the image while enhancing its visual appearance.

Anti-Aliasing textures

- Similar
- When we identify a point in the texture map we return an average of texture map around the point.
- Scaling needs to be applied so that the less the samples taken the bigger the local area where averaging is done.