CSE 167:
Introduction to Computer Graphics
Lecture #8: Textures

Jürgen P. Schulze, Ph.D.
University of California, San Diego
Fall Quarter 2010

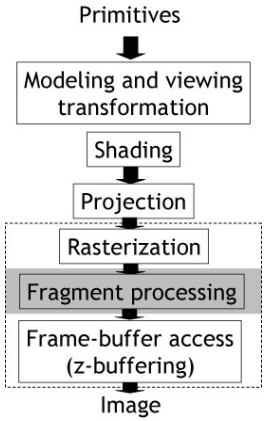
Announcements

- No new homework assignment for this Friday
- Homework assignment #4 due Friday, Oct 29
  - To be presented between 2-4pm in lab 260
- Late submissions for project #3 accepted until this Friday
- Midterm exam: Thursday, Oct 21, 2-3:20pm, WLH 2005
- Midterm tutorial: Tuesday, Oct 19, noon-1:45pm, Atkinson Hall, room 4004
  - Tutors: Jurgen and Phi
  - We will have blank index cards for everybody
- Phi's office hours on Oct 19 and 21 are cancelled

Lecture Overview

- Shader programming
  - Fragment shaders
  - GLSL
- Texturing
  - Overview
  - Texture coordinate assignment
  - Anti-aliasing

Fragment Programs



Fragment data

Interpolated vertex attributes, additional fragment attributes

From rasterizer

Uniform parameters

OpenGL state, application specified parameters

Fragment program

To fixed framebuffer access functionality (z-buffering, etc.)

Fragment color, depth

Types of Input Data

Fragment data

- Change for each execution of the fragment program
- Interpolated from vertex data during rasterization, `varying` variables

- Interpolated fragment color, texture coordinates
- Standard OpenGL fragment data accessible through predefined variables

```
varying vec4 gl_Color;
varying vec4 gl_TexCoord[ ];
```

etc.

- Note `varying` storage classifier, read-only
- User defined data possible, too

Types of Input Data

Uniform parameters

- Same as in vertex shader
- OpenGL state
- Application defined parameters
  - Use `glGetUniformLocation, glUniform*` in application

Output Variables

- Predefined outputs
  - `gl_FragColor`
  - `gl_FragDepth`
- OpenGL writes these to the frame buffer
- Result is undefined if you do not write these variables

“Hello World” Fragment Program

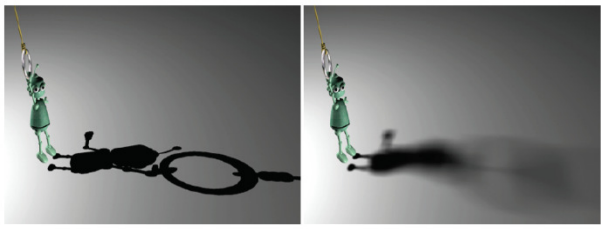
- `main()` function is executed for every fragment
- Use predefined variables
- Draws every pixel in green color

```
void main()
{
  gl_FragColor = vec4(0.0,1.0,0.0,1.0);
}
```

Examples

- Fancy per pixel shading
  - Bump mapping
  - Displacement mapping
  - Realistic reflection models
  - Cartoon shading
  - Shadows

- Most often, vertex and fragment shaders work together to achieve a desired effect



Fragment Programs

Limitations

- Cannot read frame buffer (color, depth)
- Can only write to frame buffer pixel that corresponds to fragment being processed
  - No random write access to frame buffer
- Limited number of `varying` variables passed from vertex to fragment shader
- Limited number of application-defined uniform parameters

Lecture Overview

- Shader programming
  - Fragment shaders
  - GLSL
- Texturing
  - Overview
  - Texture coordinate assignment
  - Anti-aliasing

GLSL Main Features

- Similar to C language
- `attribute, uniform, varying` storage classifiers
- Set of predefined variables
  - Access per vertex, per fragment data
  - Access OpenGL state
- Built-in vector data types, vector operations
- No pointers
- No direct access to data, variables in your C++ code

Per-pixel Diffuse Lighting

```
// Vertex shader, stored in file diffuse.vert
varying vec3 normal, lightDir;
void main()
{
  lightDir = normalize(vec3(gl_LightSource[0].position));
  normal = normalize(gl_NormalMatrix * gl_Normal);
  gl_Position = ftransform();
}
```

```
// Pixel shader, stored in file diffuse.frag
varying vec3 normal, lightDir;
void main()
{
        gl_FragColor =
       gl_LightSource[0].diffuse *
       max(dot(normalize(normal), normalize(lightDir)),0.0) *
       gl_FrontMaterial.diffuse;
}
```

GLSL Quick Reference Guide

OpenGL® Shading Language (GLSL) Quick Reference Guide

Describes GLSL version 1.10, as included in OpenGL v2.0, and specified by "The OpenGL® Shading Language", version 1.10.59. Section and page numbers refer to that version of the spec.

DATA TYPES (4.1 p16)

float, vec2, vec3, vec4
int, ivec2, ivec3, ivec4
bool, bvec2, bvec3, bvec4
mat2, mat3, mat4
void
sampler1D, sampler2D, sampler3D
samplerCube
sampler1DShadow, sampler2DShadow

DATA TYPE QUALIFIERS (4.3 p22)

global variable declarations:

uniform	input to Vertex and Fragment shader from OpenGL or application (READ-ONLY)
attribute	input per-vertex to Vertex shader from OpenGL or application (READ-ONLY)
varying	output from Vertex shader (READ/WRITE), interpolated, then input to Fragment shader (READ-ONLY)
const	compile-time constant (READ-ONLY)

function parameters:

in	value initialized on entry, not copied or return (default)
out	copied out on return, but not initialized
inout	value initialized on entry, and copied out on return
const	constant function input

VECTOR COMPONENTS (5.5 p 30)

component names may not be mixed across sets

x, y, z, w
r, g, b, a
s, t, p, q

PREPROCESSOR (3.3 p9)

#
#define
#undef
#if
#ifdef
#ifndef
#else
#elif
#endif
#error
#pragma
#line

__LINE__
__FILE__
__VERSION__

GLSL version declaration and extensions protocol:

#version
default is "#version 110" (3.3 p11)
#extension {*name* | all} : {require | enable | warn | disable}
default is "#extension all : disable" (3.3 p11)

BUILT-IN FUNCTIONS

Key:
vec = vec2 | vec3 | vec4
mat = mat2 | mat3 | mat4
ivec = ivec2 | ivec3 | ivec4
bvec = bvec2 | bvec3 | bvec4
genType = float | vec2 | vec3 | vec4

Angle and Trigonometry Functions (8.1 p51)
genType sin(genType)
genType cos(genType)
genType tan(genType)

genType asin(genType)
genType acos(genType)
genType atan(genType, genType)
genType atan(genType)

genType radians(genType)
genType degrees(genType)

Exponential Functions (8.2 p52)
genType pow(genType, genType)
genType exp(genType)
genType log(genType)
genType exp2(genType)
genType log2(genType)
genType sqrt(genType)
genType inversesqrt(genType)

Common Functions (8.3 p52)
genType abs(genType)
genType ceil(genType)
genType clamp(genType, genType, genType)
genType clamp(genType, float, float)
genType floor(genType)
genType fract(genType)
genType max(genType, genType)
genType max(genType, float)
genType min(genType, genType)
genType min(genType, float)
genType mix(genType, genType, genType)
genType mix(genType, genType, float)
genType mod(genType, genType)
genType mod(genType, float)
genType sign(genType)
genType smoothstep(genType, genType, genType)
genType smoothstep(float, float, genType)
genType step(genType, genType)
genType step(float, genType)

Geometric Functions (8.4 p54)
vec4 ftransform() *Vertex ONLY*
vec3 cross(vec3, vec3)
float distance(genType, genType)
float dot(genType, genType)
genType faceforward(genType *V*, genType *I*, genType *N*)
float length(genType)
genType normalize(genType)
genType reflect(genType *I*, genType *N*)
genType refract(genType *I*, genType *N*, float *eta*)

Fragment Processing Functions (8.8 p58) *Fragment ONLY*
genType dFdx(genType)
genType dFdy(genType)
genType fwidth(genType)

Matrix Functions (8.5 p55)
mat matrixCompMult(mat, mat)

Vector Relational Functions (8.6 p55)
bool all(bvec)
bool any(bvec)
bvec equal(vec, vec)
bvec equal(ivec, ivec)
bvec equal(bvec, bvec)
bvec greaterThan(vec, vec)
bvec greaterThan(ivec, ivec)
bvec greaterThanEqual(vec, vec)
bvec greaterThanEqual(ivec, ivec)
bvec lessThan(vec, vec)
bvec lessThan(ivec, ivec)
bvec lessThanEqual(vec, vec)
bvec lessThanEqual(ivec, ivec)
bvec not(bvec)
bvec notEqual(vec, vec)
bvec notEqual(vec, ivec)
bvec notEqual(bvec, bvec)

Texture Lookup Functions (8.7 p56)
Optional bias term is Fragment ONLY
vec4 texture1D(sampler1D, float [,float *bias*])
vec4 texture1DProj(sampler1D, vec2 [,float *bias*])
vec4 texture1DProj(sampler1D, vec4 [,float *bias*])

vec4 texture2D(sampler2D, vec2 [,float *bias*])
vec4 texture2DProj(sampler2D, vec3 [,float *bias*])
vec4 texture2DProj(sampler2D, vec4 [,float *bias*])

vec4 texture3D(sampler3D, vec3 [,float *bias*])
vec4 texture3DProj(sampler3D, vec4 [,float *bias*])

vec4 textureCube(samplerCube, vec3 [,float *bias*])

vec4 shadow1D(sampler1DShadow, vec3 [,float *bias*])
vec4 shadow2D(sampler2DShadow, vec3 [,float *bias*])
vec4 shadow1DProj(sampler1DShadow, vec4 [,float *bias*])
vec4 shadow2DProj(sampler2DShadow, vec4 [,float *bias*])

Texture Lookup Functions with LOD (8.7 p56)
Vertex ONLY; ensure GL_MAX_VERTEX_TEXTURE_IMAGE_UNITS > 0
vec4 texture1DLod(sampler1D, float, float *lod*)
vec4 texture1DProjLod(sampler1D, vec2, float *lod*)
vec4 texture1DProjLod(sampler1D, vec4, float *lod*)

vec4 texture2DLod(sampler2D, vec2, float *lod*)
vec4 texture2DProjLod(sampler2D, vec3, float *lod*)
vec4 texture2DProjLod(sampler2D, vec4, float *lod*)
vec4 texture3DProjLod(sampler3D, vec4, float *lod*)

vec4 textureCubeLod(samplerCube, vec3, float *lod*)

vec4 shadow1DLod(sampler1DShadow, vec3, float *lod*)
vec4 shadow2DLod(sampler2DShadow, vec3, float *lod*)
vec4 shadow1DProjLod(sampler1DShadow, vec4, float *lod*)
vec4 shadow2DProjLod(sampler2DShadow, vec4, float *lod*)

Noise Functions (8.9 p60)
float noise1(genType)
vec2 noise2(genType)
vec3 noise3(genType)
vec4 noise4(genType)

http://www.opengl.org/sdk/libs/OpenSceneGraph/glsl_quickref.pdf

GLSL Quick Reference Guide

VERTEX SHADER VARIABLES

Special Output Variables (7.1 p42) *access=RW*
```
vec4  gl_Position;      shader must write
float gl_PointSize;     enable GL_VERTEX_PROGRAM_POINT_SIZE
vec4  gl_ClipVertex;
```

Attribute Inputs (7.3 p44) *access=RO*
```
attribute vec4  gl_Vertex;
attribute vec3  gl_Normal;
attribute vec4  gl_Color;
attribute vec4  gl_SecondaryColor;
attribute vec4  gl_MultiTexCoord0;
attribute vec4  gl_MultiTexCoord1;
attribute vec4  gl_MultiTexCoord2;
attribute vec4  gl_MultiTexCoord3;
attribute vec4  gl_MultiTexCoord4;
attribute vec4  gl_MultiTexCoord5;
attribute vec4  gl_MultiTexCoord6;
attribute vec4  gl_MultiTexCoord7;
attribute float gl_FogCoord;
```

Varying Outputs (7.6 p48) *access=RW*
```
varying vec4  gl_FrontColor;
varying vec4  gl_BackColor;  enable GL_VERTEX_PROGRAM_TWO_SIDE
varying vec4  gl_FrontSecondaryColor;
varying vec4  gl_BackSecondaryColor;
varying vec4  gl_TexCoord[ ];     MAX=gl_MaxTextureCoords
varying float gl_FogFragCoord;
```

FRAGMENT SHADER VARIABLES

Special Output Variables (7.2 p43) *access=RW*
```
vec4  gl_FragColor;
vec4  gl_FragData[gl_MaxDrawBuffers];
float gl_FragDepth;     DEFAULT=glFragCoord.z
```

Varying Inputs (7.6 p48) *access=RO*
```
varying vec4  gl_Color;
varying vec4  gl_SecondaryColor;
varying vec4  gl_TexCoord[ ];     MAX=gl_MaxTextureCoords
varying float gl_FogFragCoord;
```

Special Input Variables (7.2 p43) *access=RO*
```
vec4  gl_FragCoord;          pixel coordinates
bool  gl_FrontFacing;
```

BUILT-IN CONSTANTS (7.4 p44)
```
const int gl_MaxVertexUniformComponents;
const int gl_MaxFragmentUniformComponents;
const int gl_MaxVertexAttribs;
const int gl_MaxVaryingFloats;
const int gl_MaxDrawBuffers;
const int gl_MaxTextureCoords;
const int gl_MaxTextureUnits;
const int gl_MaxTextureImageUnits;
const int gl_MaxVertexTextureImageUnits;
const int gl_MaxCombinedTextureImageUnits;
const int gl_MaxLights;
const int gl_MaxClipPlanes;
```

BUILT-IN UNIFORMs (7.5 p45) *access=RO*
```
uniform mat4 gl_ModelViewMatrix;
uniform mat4 gl_ModelViewProjectionMatrix;
uniform mat4 gl_ProjectionMatrix;
uniform mat4 gl_TextureMatrix[gl_MaxTextureCoords];

uniform mat4 gl_ModelViewMatrixInverse;
uniform mat4 gl_ModelViewProjectionMatrixInverse;
uniform mat4 gl_ProjectionMatrixInverse;
uniform mat4 gl_TextureMatrixInverse[gl_MaxTextureCoords];

uniform mat4 gl_ModelViewMatrixTranspose;
uniform mat4 gl_ModelViewProjectionMatrixTranspose;
uniform mat4 gl_ProjectionMatrixTranspose;
uniform mat4 gl_TextureMatrixTranspose[gl_MaxTextureCoords];

uniform mat4 gl_ModelViewMatrixInverseTranspose;
uniform mat4 gl_ModelViewProjectionMatrixInverseTranspose;
uniform mat4 gl_ProjectionMatrixInverseTranspose;
uniform mat4 gl_TextureMatrixInverseTranspose[gl_MaxTextureCoords];

uniform mat3 gl_NormalMatrix;
uniform float gl_NormalScale;

struct gl_DepthRangeParameters {
   float near;
   float far;
   float diff;
};
uniform gl_DepthRangeParameters gl_DepthRange;

struct gl_FogParameters {
   vec4 color;
   float density;
   float start;
   float end;
   float scale;
};
uniform gl_FogParameters gl_Fog;

struct gl_LightSourceParameters {
   vec4 ambient;
   vec4 diffuse;
   vec4 specular;
   vec4 position;
   vec4 halfVector;
   vec3 spotDirection;
   float spotExponent;
   float spotCutoff;
   float spotCosCutoff;
   float constantAttenuation;
   float linearAttenuation;
   float quadraticAttenuation;
};
uniform gl_LightSourceParameters gl_LightSource[gl_MaxLights];

struct gl_LightModelParameters {
   vec4 ambient;
};
uniform gl_LightModelParameters gl_LightModel;

struct gl_LightModelProducts {
   vec4 sceneColor;
};
uniform gl_LightModelProducts gl_FrontLightModelProduct;
uniform gl_LightModelProducts gl_BackLightModelProduct;

struct gl_LightProducts {
   vec4 ambient;
   vec4 diffuse;
   vec4 specular;
};
uniform gl_LightProducts gl_FrontLightProduct[gl_MaxLights];
uniform gl_LightProducts gl_BackLightProduct[gl_MaxLights];

struct gl_MaterialParameters {
   vec4 emission;
   vec4 ambient;
   vec4 diffuse;
   vec4 specular;
   float shininess;
};
uniform gl_MaterialParameters gl_FrontMaterial;
uniform gl_MaterialParameters gl_BackMaterial;

struct gl_PointParameters {
   float size;
   float sizeMin;
   float sizeMax;
   float fadeThresholdSize;
   float distanceConstantAttenuation;
   float distanceLinearAttenuation;
   float distanceQuadraticAttenuation;
};
uniform gl_PointParameters gl_Point;

uniform vec4 gl_TextureEnvColor[gl_MaxTextureUnits];  (1)

uniform vec4 gl_ClipPlane[gl_MaxClipPlanes];

uniform vec4 gl_EyePlaneS[gl_MaxTextureCoords];
uniform vec4 gl_EyePlaneT[gl_MaxTextureCoords];
uniform vec4 gl_EyePlaneR[gl_MaxTextureCoords];
uniform vec4 gl_EyePlaneQ[gl_MaxTextureCoords];

uniform vec4 gl_ObjectPlaneS[gl_MaxTextureCoords];
uniform vec4 gl_ObjectPlaneT[gl_MaxTextureCoords];
uniform vec4 gl_ObjectPlaneR[gl_MaxTextureCoords];
uniform vec4 gl_ObjectPlaneQ[gl_MaxTextureCoords];
```

OpenSceneGraph Preset Uniforms
as of OSG 1.0
```
int    osg_FrameNumber;
float  osg_FrameTime;
float  osg_DeltaFrameTime;
mat4   osg_ViewMatrix;
mat4   osg_ViewMatrixInverse;
```


Notes
1. Corrects a typo in the OpenGL 2.0 specification.

http://www.opengl.org/sdk/libs/OpenSceneGraph/glsl_quickref.pdf

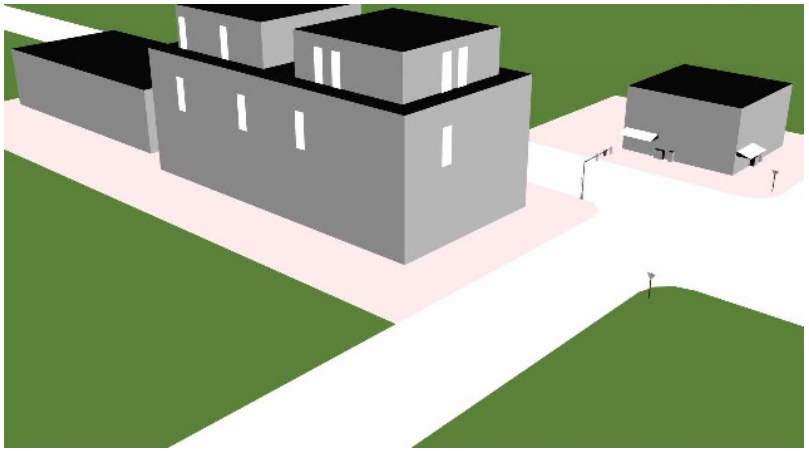
Tutorials and Documentation

- OpenGL and GLSL specifications
  http://www.opengl.org/documentation/specs/
- GLSL tutorials
  http://www.lighthouse3d.com/opengl/glsl/
  http://www.clockworkcoders.com/oglsl/tutorials.html
- OpenGL Programming Guide (Red Book)
- OpenGL Shading Language (Orange Book)

Lecture Overview

- Shader programming
  - Fragment shaders
  - GLSL
- Texturing
  - Overview
  - Texture coordinate assignment
  - Anti-aliasing

Large Triangles



Pros:

- Often sufficient for simple geometry
- Fast to render

Cons:

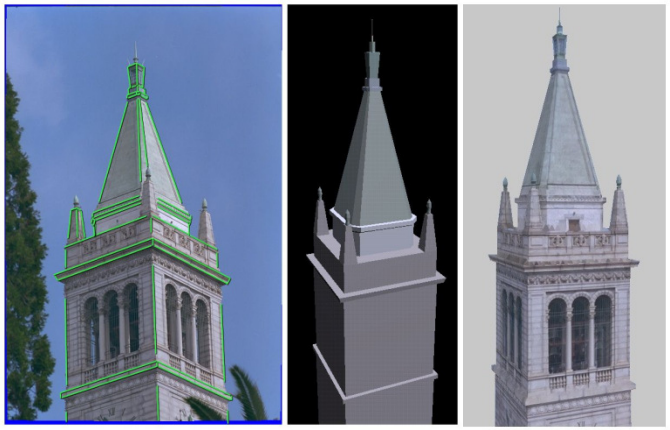
- Per vertex colors look bad
- Need more interesting surfaces
  - Detailed color variation, small scale bumps, roughness

Texture Mapping

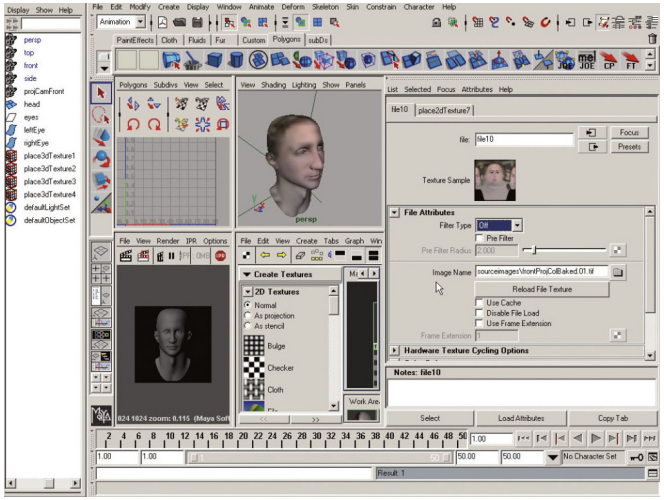
- Attach textures (images) onto surfaces
- Same triangle count, much more realistic appearance

Texture Sources

- Take photographs
- Paint directly on surfaces with a 3D modeling program (Maya, 3ds Max, Blender, etc.)
- Use existing images from disk



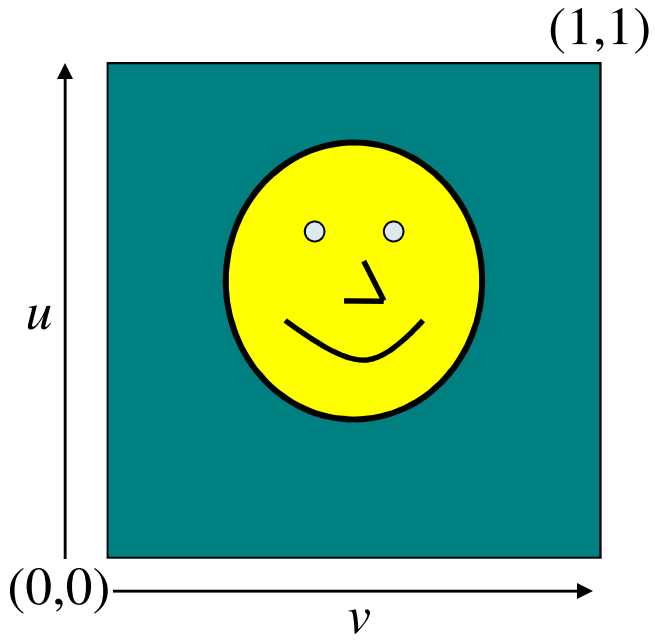
Images by Paul Debevec



Texture painting in Maya

Texture Mapping

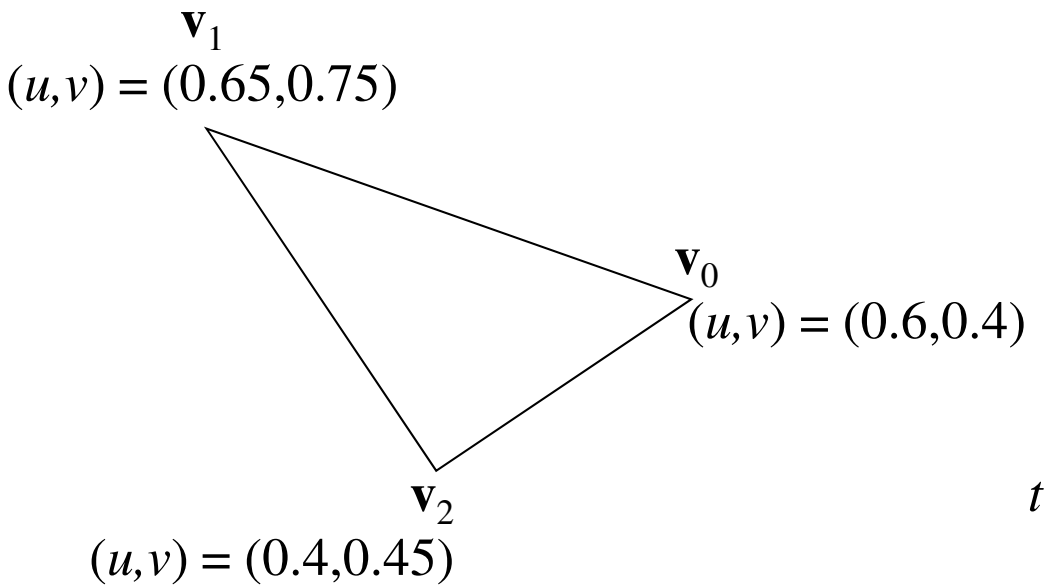
- Goal: assign locations in texture to locations on 3D geometry
- Introduce texture space
  - Texture pixels (texels) have texture coordinates (u,v)
- Convention
  - Bottom left corner of texture is at $(u,v) = (0,0)$
  - Top right corner is at $(u,v) = (1,1)$



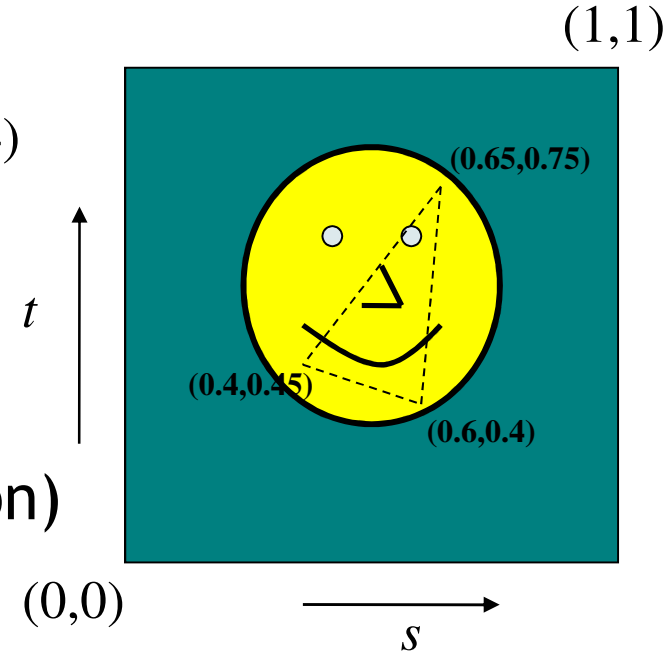
Texture space

Texture Mapping

- Store texture coordinates at each triangle vertex



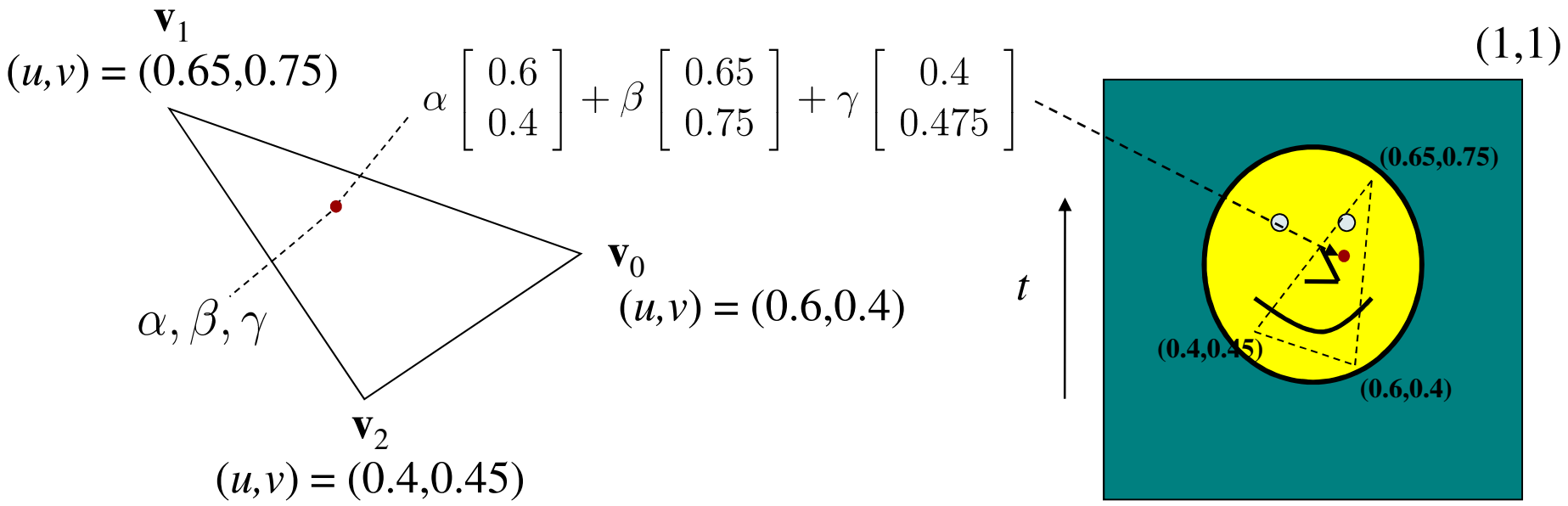
Triangle (in any space before projection)



Texture space

Texture Mapping

- Each point on triangle has barycentric coordinates α, β, γ
- Use barycentric coordinates to interpolate texture coordinates

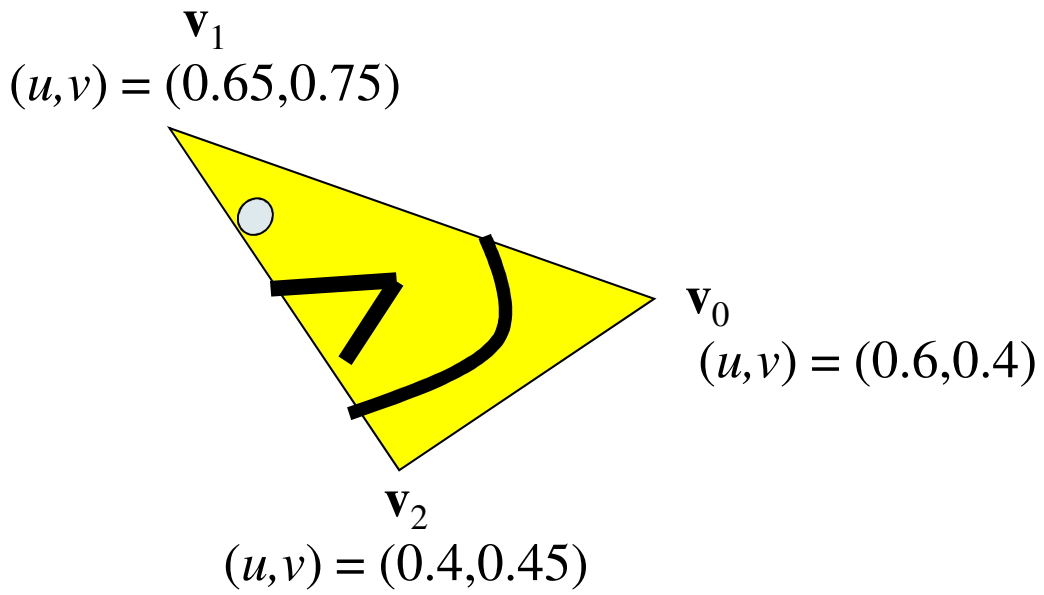


Triangle (in any space before projection)

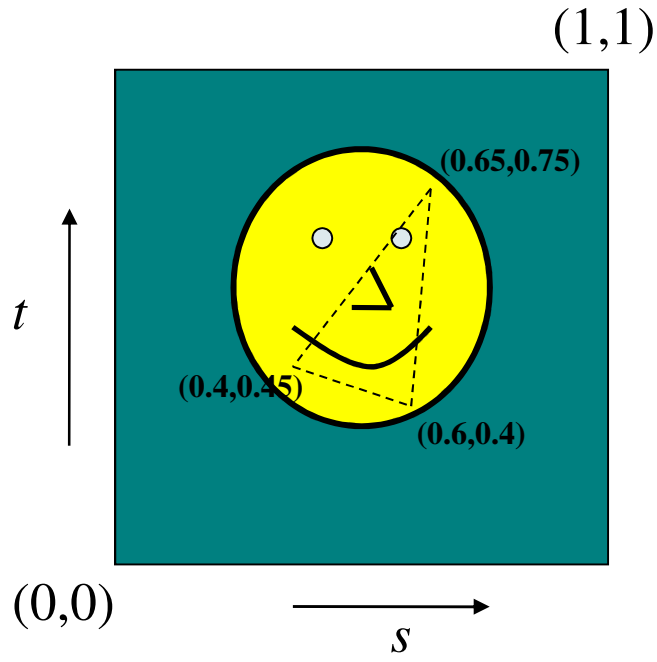
Texture space

Texture Mapping

- Each point on triangle has corresponding point in texture



Triangle (in any space before projection)

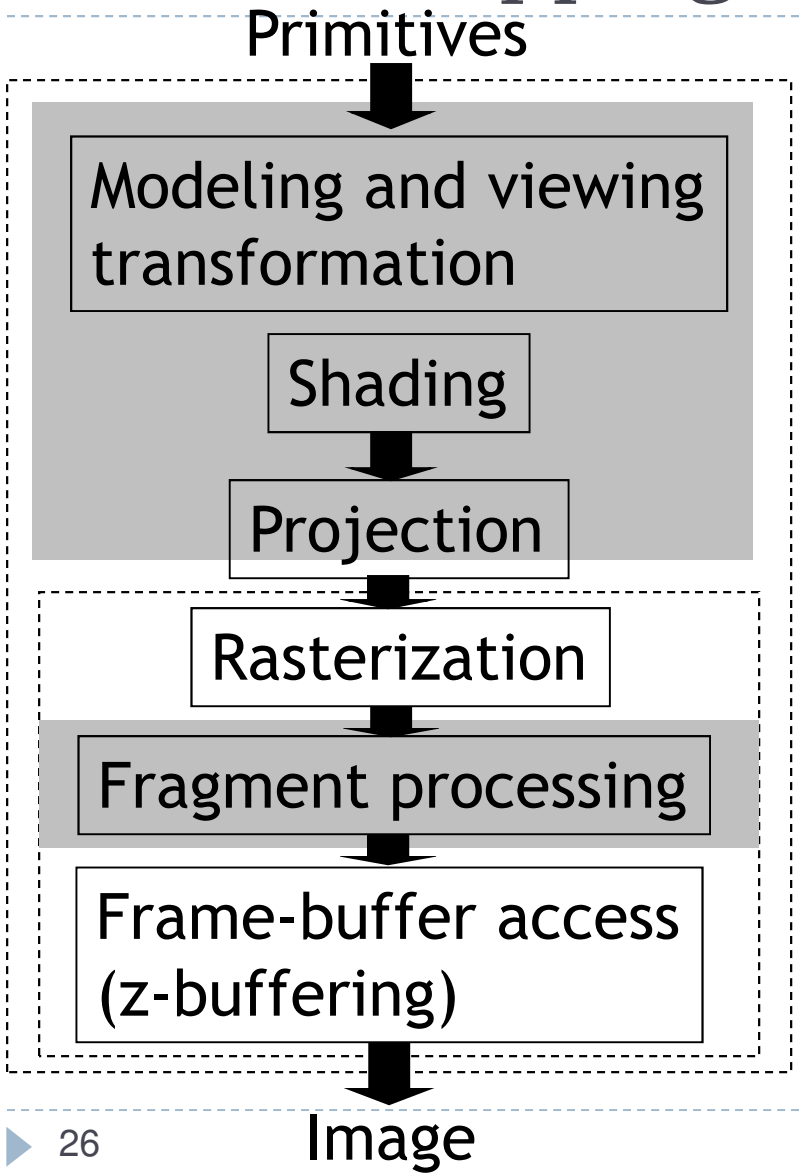


Texture space

Rendering

- Given
  - Texture coordinates at each vertex
  - Texture image
- At each pixel, use barycentric coordinates to interpolate texture coordinates
- Look up corresponding texel
- Paint current pixel with texel color
- All computations are done on the GPU

Texture Mapping

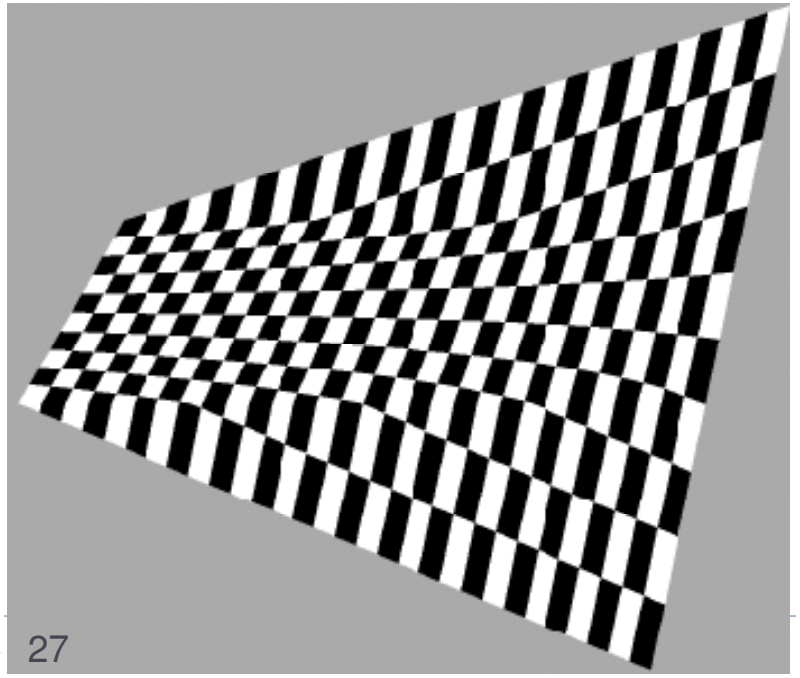


Includes texture mapping

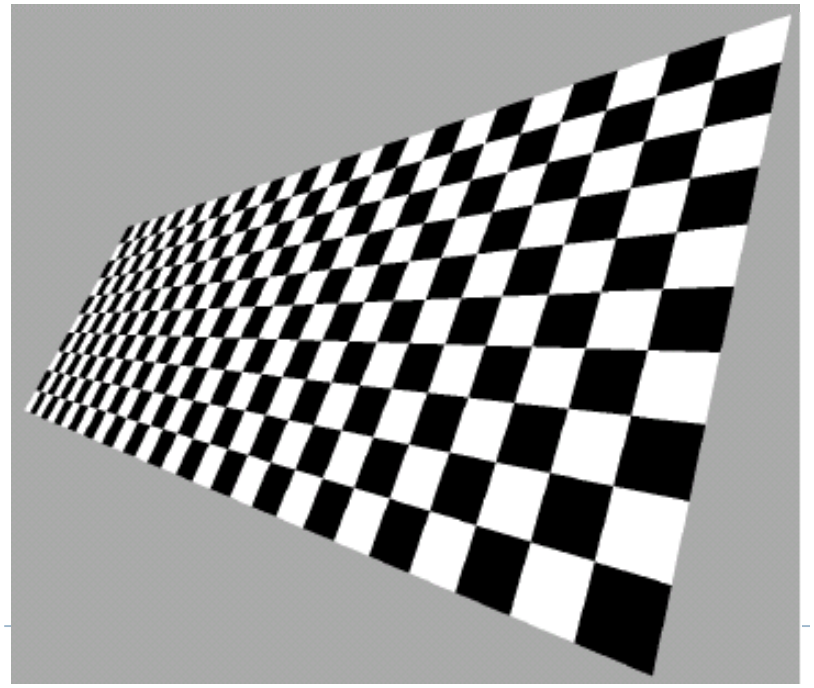
Rendering

- Linear interpolation in image space does not correspond to linear interpolation in 3D
- Need to do perspectively correct interpolation!

Linear interpolation in image coordinates



Perspectively correct interpolation

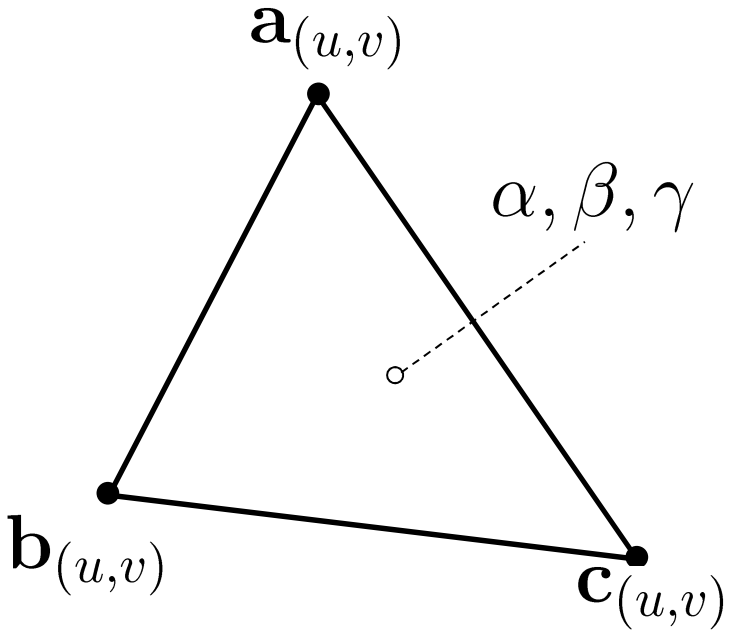


Perspectively Correct Interpolation

- Point in image space with barycentric coordinates α, β, γ
- Triangle vertices with homogeneous coordinates

1. a_w, b_w, c_w
2. $\frac{1}{w} = \alpha\frac{1}{a_w} + \beta\frac{1}{b_w} + \gamma\frac{1}{c_w}$
3. $\frac{u}{w} = \alpha\frac{\mathbf{a}_u}{a_w} + \beta\frac{\mathbf{b}_u}{b_w} + \gamma\frac{\mathbf{c}_u}{c_w}$

$$u = \frac{u}{w} \Big/ \frac{1}{w}$$

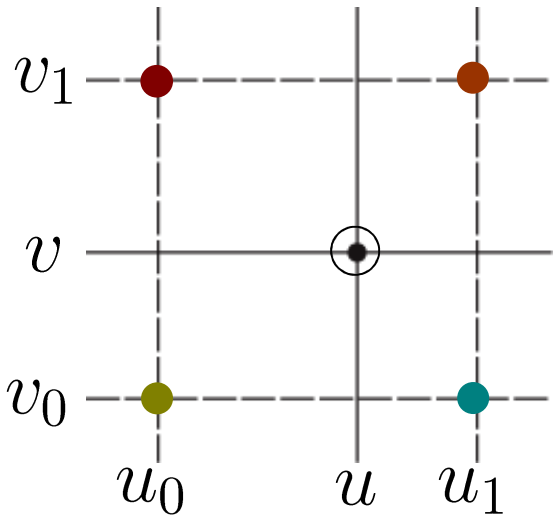


- Same for v texture coordinate

Texture Look-Up

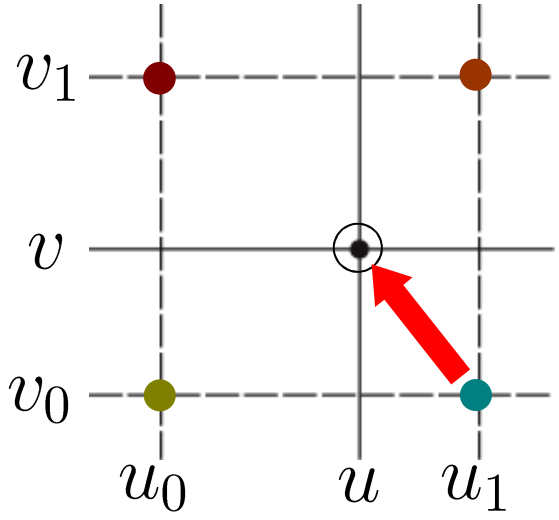
- Given interpolated texture coordinates (*u*, *v*) at current pixel
- Closest four texels in texture space are at

  $(u_0, v_0), (u_1, v_0), (u_1, v_0), (u_1, v_1)$
- How to compute pixel color?



Nearest-Neighbor Interpolation

- Use color of closest texel



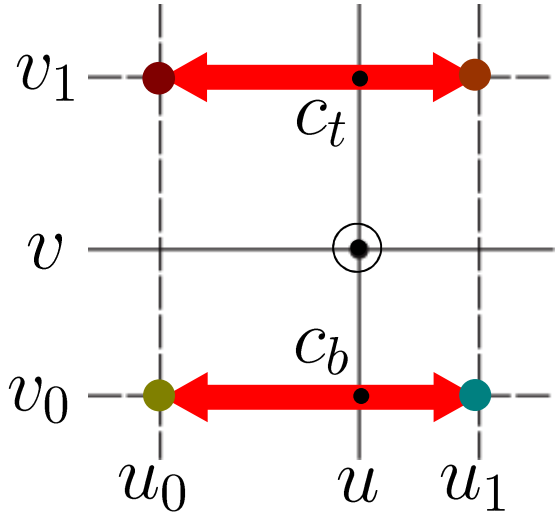
- Simple, but low quality and aliasing

Bilinear Interpolation

1. Linear interpolation horizontally

$$w_u = \frac{u - u_0}{u_1 - u_0}$$

$$\begin{aligned} c_b &= tex(u_0, v_0)(1 - w_u) + tex(u_1, v_0)w_u \\ c_t &= tex(u_0, v_1)(1 - w_u) + tex(u_1, v_1)w_u \end{aligned}$$



Bilinear Interpolation

1. Linear interpolation horizontally

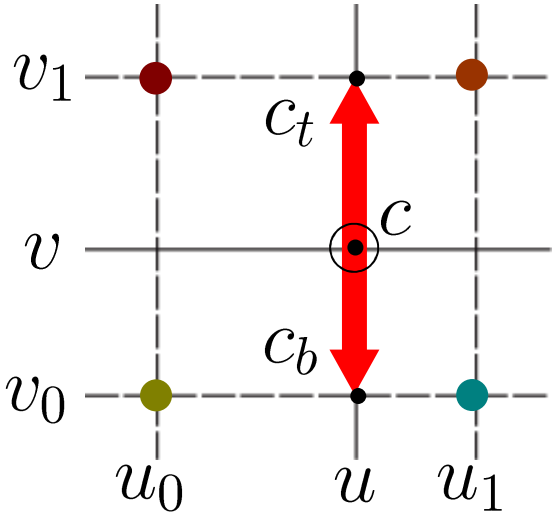
$$w_u = \frac{u - u_0}{u_1 - u_0}$$

$$\begin{aligned} c_b &= tex(u_0, v_0)(1 - w_u) + tex(u_1, v_0)w_u \\ c_t &= tex(u_0, v_1)(1 - w_u) + tex(u_1, v_1)w_u \end{aligned}$$

2. Linear interpolation vertically

$$w_v = \frac{v - v_0}{v_1 - v_0}$$

$$c = c_b(1 - w_v) + c_t w_v$$



Basic Shaders for Texturing

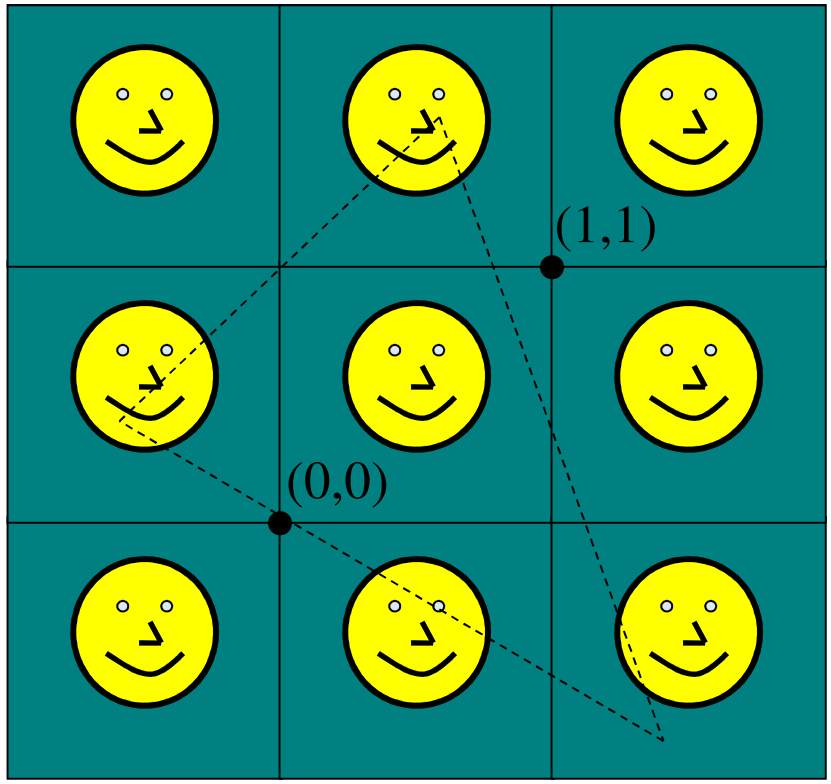
```
// Need to initialize texture using OpenGL API calls.
// See base code.

// Vertex shader
void main()
{
  gl_Position = ftransform();
}

// Fragment shader
uniform sampler2D tex;
void main()
{
  gl_FragColor = texture2D(tex, gl_TexCoord[0].st);
}
```

Tiling

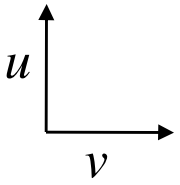
- Image exists from $[0,0]$x$[1,1]$ in texture space
- (u,v) texture coordinates may go outside that range
- *Tiling* and *wrapping* rules for out-of-range coordinates

Tiling

- Repeat the texture
  - Seams, unless the texture lines up on the edges

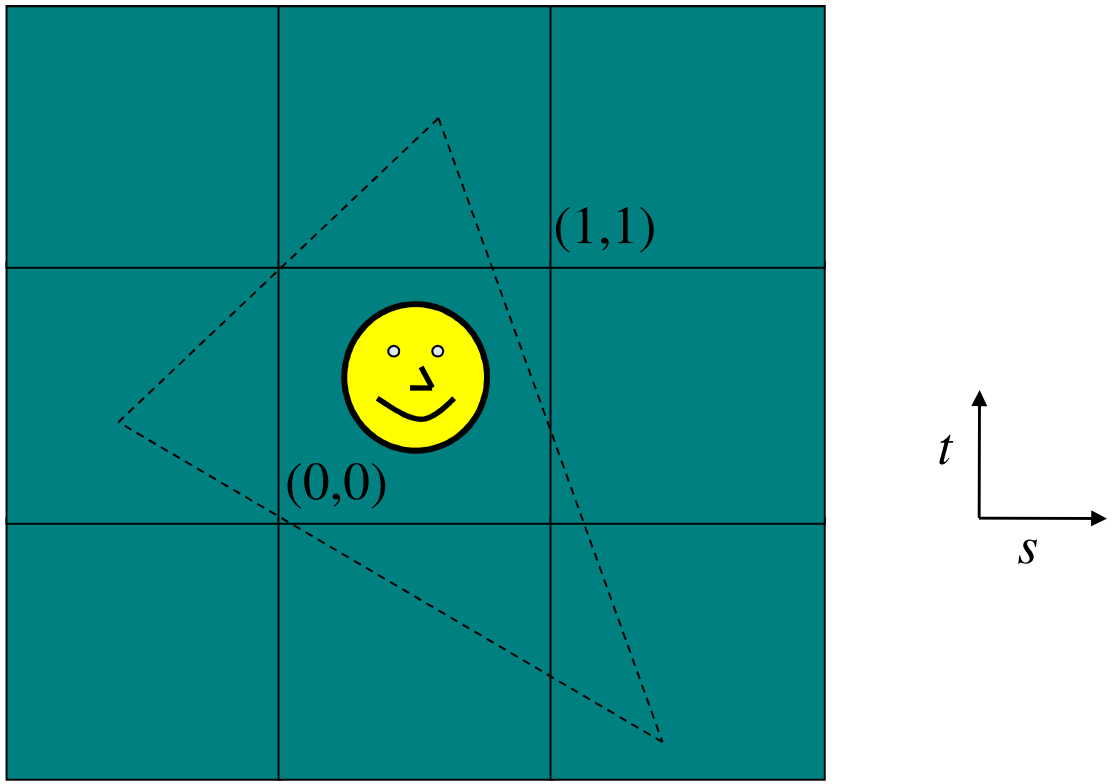


Texture Space



Clamping

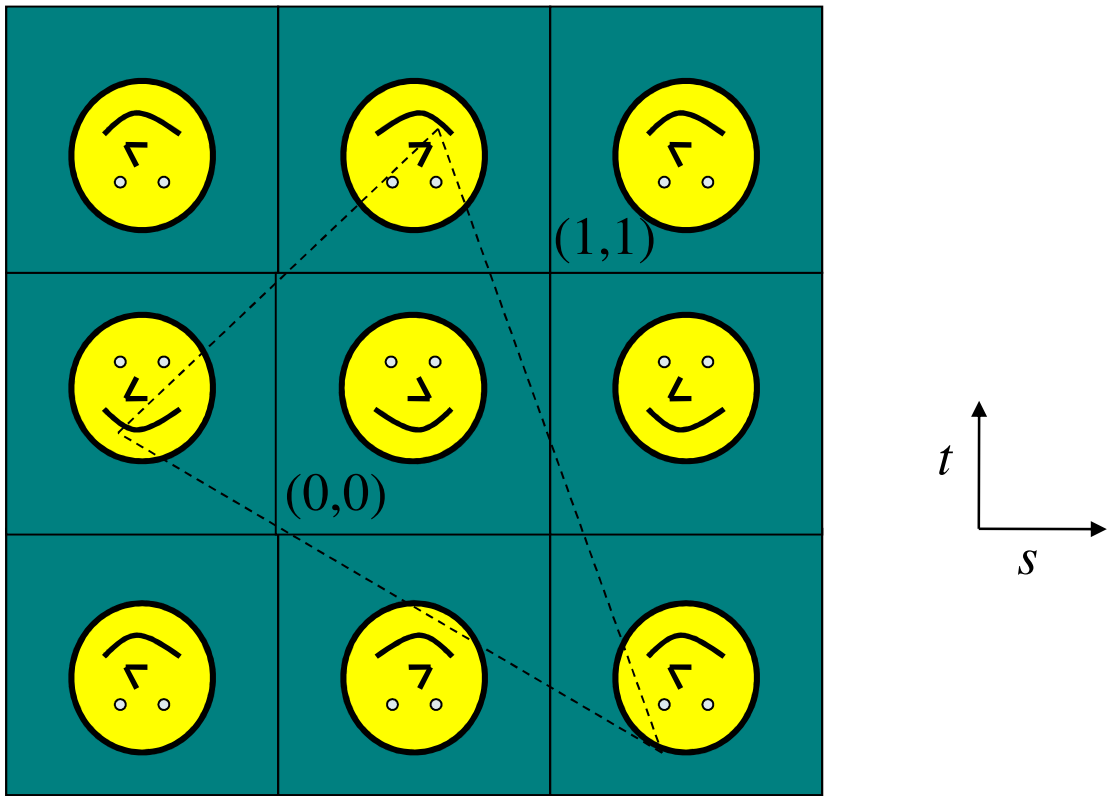
- Use the edge value everywhere outside the data
- Or, ignore the texture outside 0-1



Texture Space

Mirroring

- Flip left-to-right and top-to-bottom
  - All the edges line up



Texture Space

Lecture Overview

- Shader programming
  - Fragment shaders
  - GLSL
- Texturing
  - Overview
  - Texture coordinate assignment
  - Anti-aliasing

Texture Coordinate Assignment

Surface parameterization

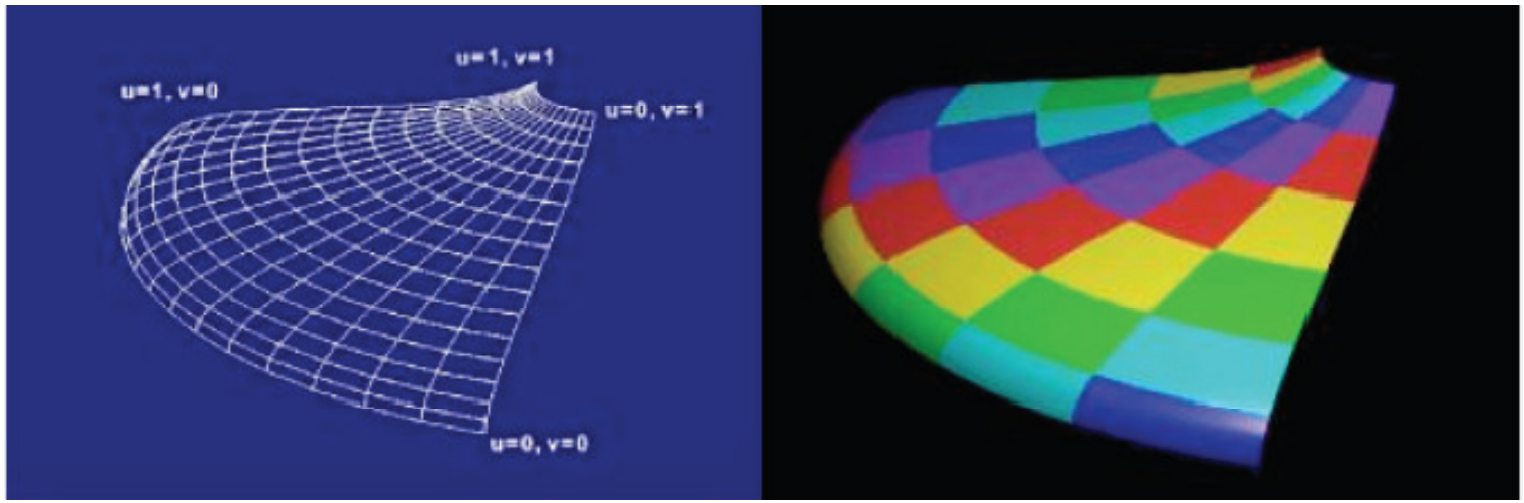
- Mapping between 3D positions on surface and 2D texture coordinates
  - In practice, defined by texture coordinates of triangle vertices
- Various options to establish a parameterization
  - Parametric mapping
  - Orthographic mapping
  - Projective mapping
  - Spherical mapping
  - Cylindrical mapping
  - Skin mapping

Parametric Mapping

- Surface given by parametric functions

$$x = f(u,v) \quad y = f(u,v) \quad z = f(u,v)$$

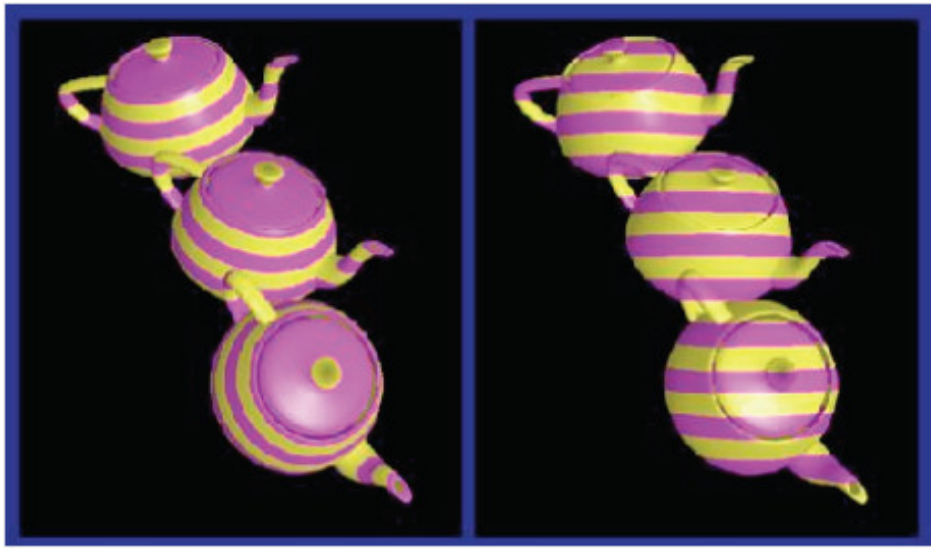
- Very common in CAD
- Use (u,v) parameters as texture coordinates



Orthographic Mapping

- Use linear transformation of object's xyz coordinates
- For example

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

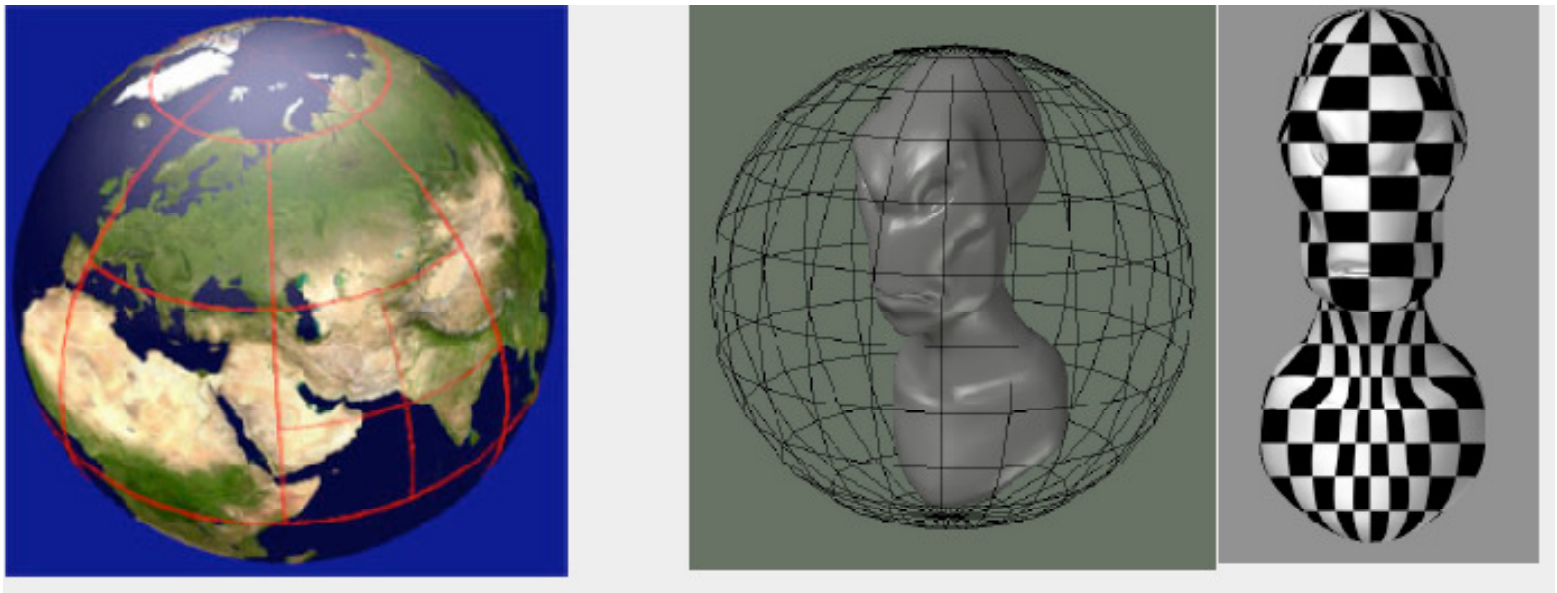


Projective Mapping

- Use perspective projection of xyz coordinates
  - OpenGL provides GL_TEXTURE matrix to apply perspective projection on texture coordinates
- Useful to achieve "fake" lighting effects

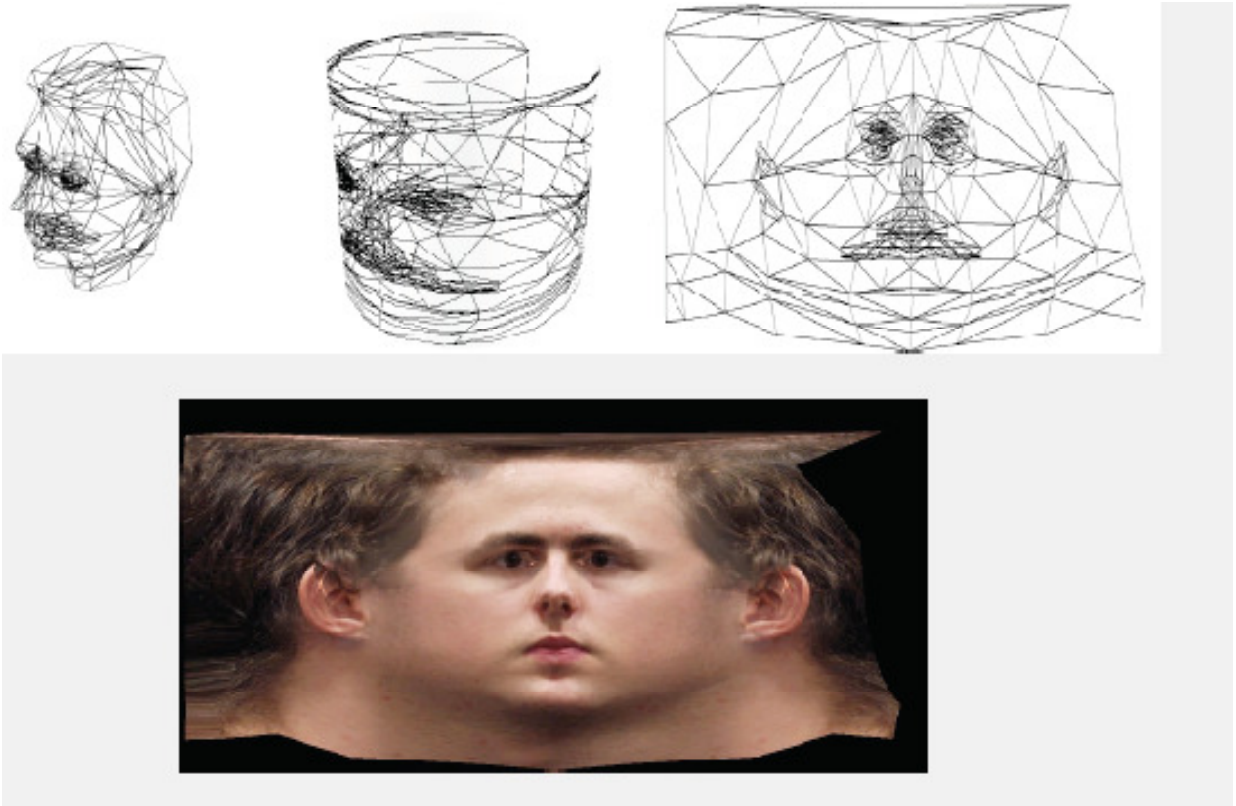
Spherical Mapping

- Use, e.g., spherical coordinates for sphere
- Place object in sphere
- "shrink-wrap" sphere to object



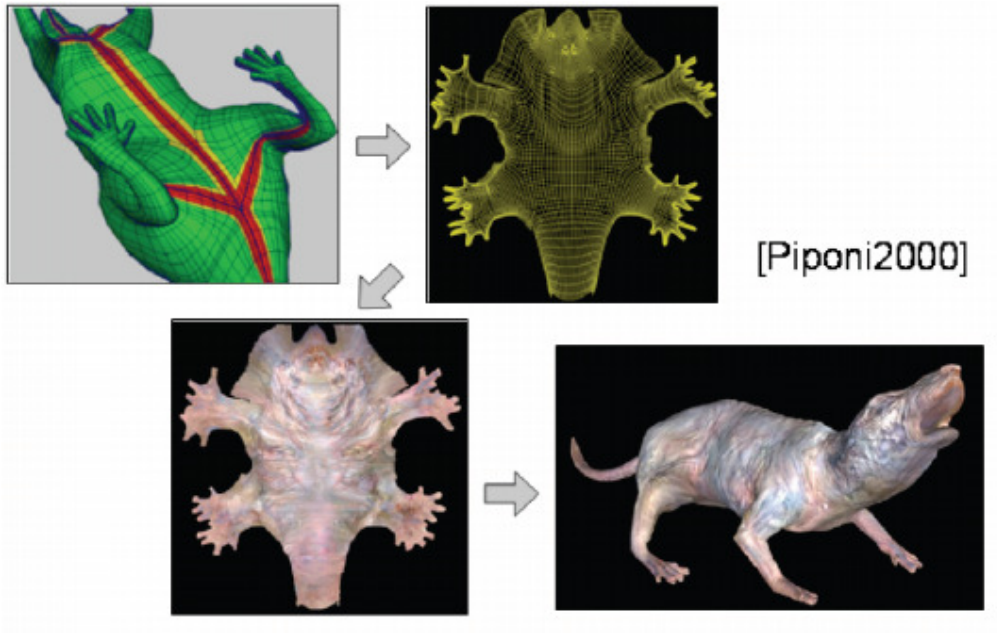
Cylindrical Mapping

- Similar as spherical mapping, but with cylilnder
- Useful for faces



Skin Mapping

- Complex technique to unfold surface onto plane
- Preserve area and angle
- Sophisticated mathematics

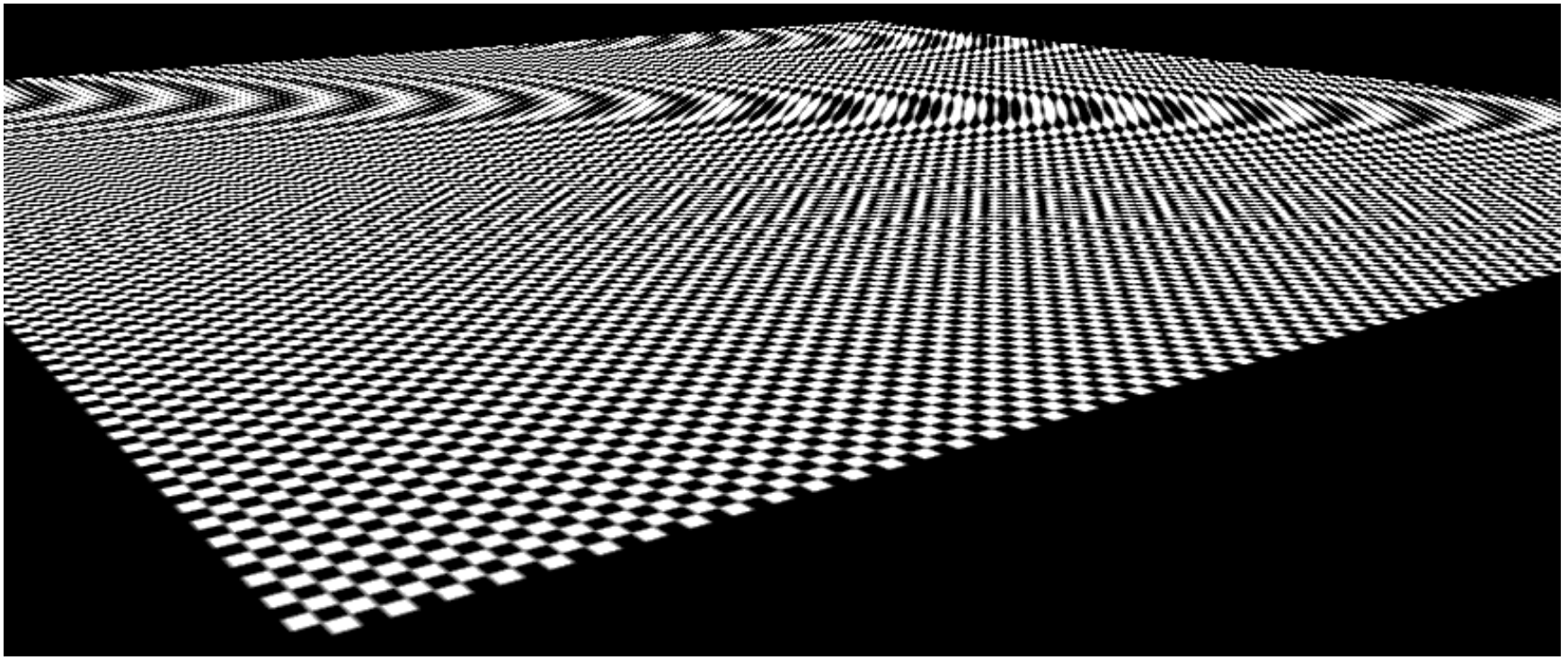


Lecture Overview

- Shader programming
  - Fragment shaders
  - GLSL
- Texturing
  - Overview
  - Texture coordinate assignment
  - Anti-aliasing

Example for Aliasing

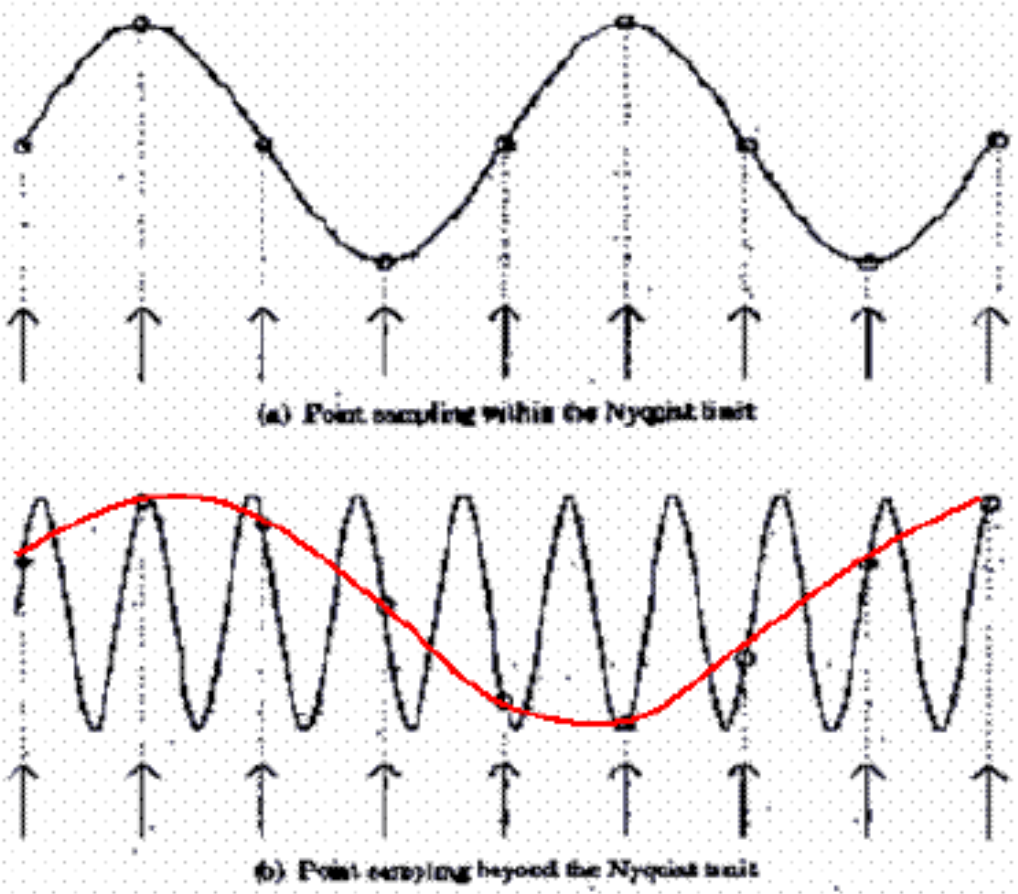
- What causes this?



Aliasing

Sufficiently sampled, no aliasing

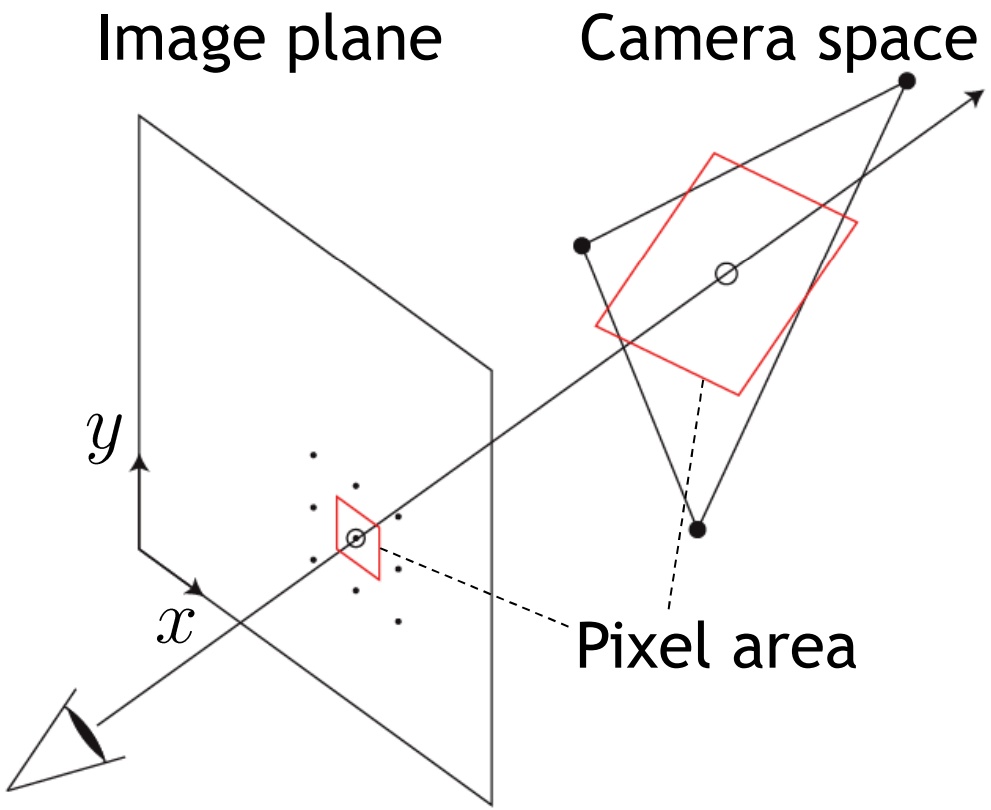
Insufficiently sampled, aliasing



High frequencies in the input data appear as low frequencies in the sampled signal

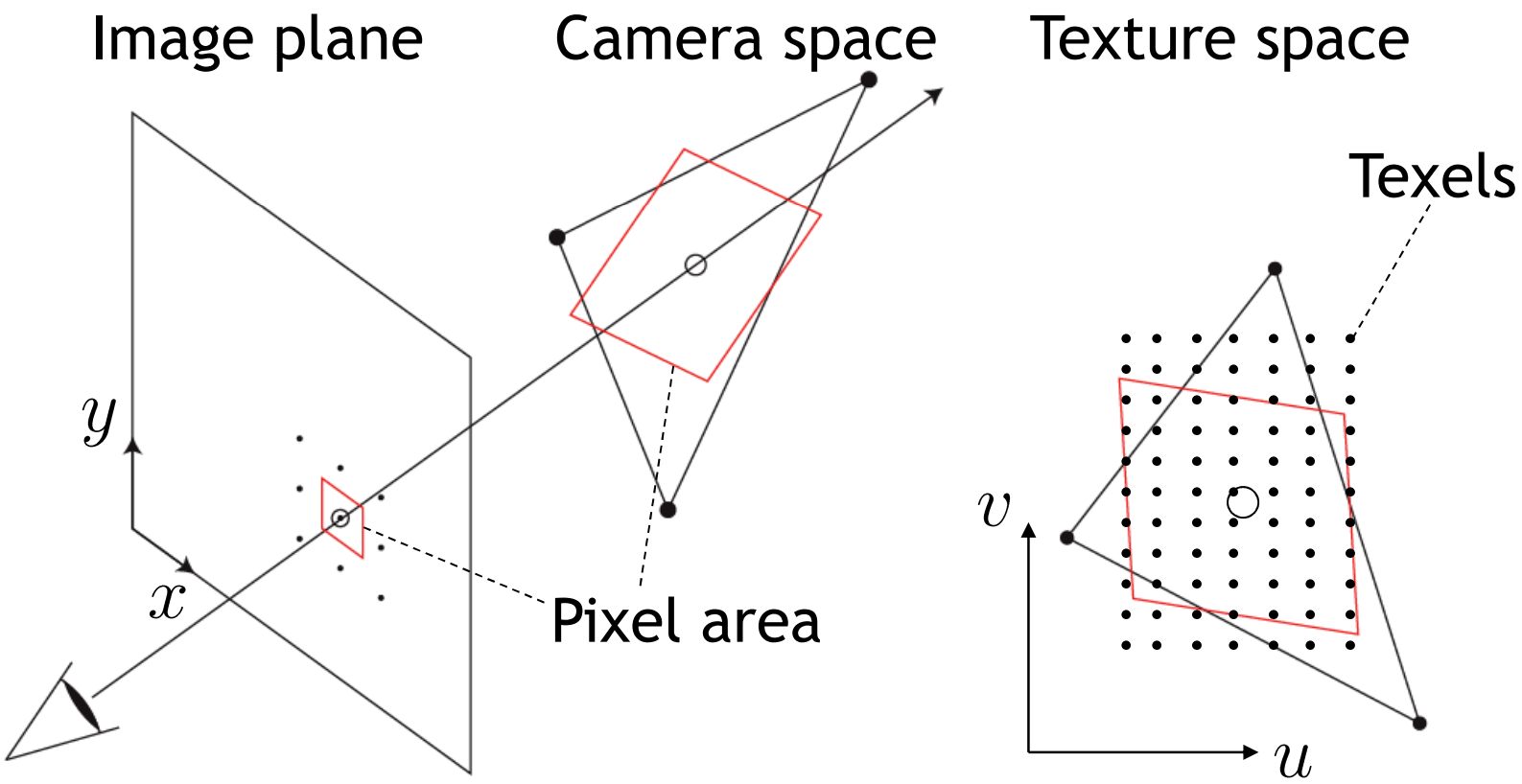
Antialiasing: Intuition

- Pixel may cover large area on triangle in camera space



Antialiasing: Intuition

- Pixel may cover large area on triangle in camera space
- Corresponds to many texels in texture space
- Need to compute average



Antialiasing: Mathematics

- Pixels are samples, not little squares
  http://www.alvyray.com/memos/6_pixel.pdf
- Use frequency analysis to explain sampling artifacts
  - Fourier transforms
- Antialiasing is achieved through low-pass filtering
- For more information:
  - http://www.cs.cmu.edu/~ph/texfund/texfund.pdf
  - Glassner: Principles of digital image synthesis

Antialiasing Using Mip-Maps

- Averaging over texels is expensive
  - Many texels as objects get smaller
  - Large memory access and compuation cost
- Precompute filtered (averaged) textures
  - Mip-maps
- Practical solution to aliasing problem
  - Fast and simple
  - Available in OpenGL, implemented in GPUs
  - Reasonable quality

Mipmaps

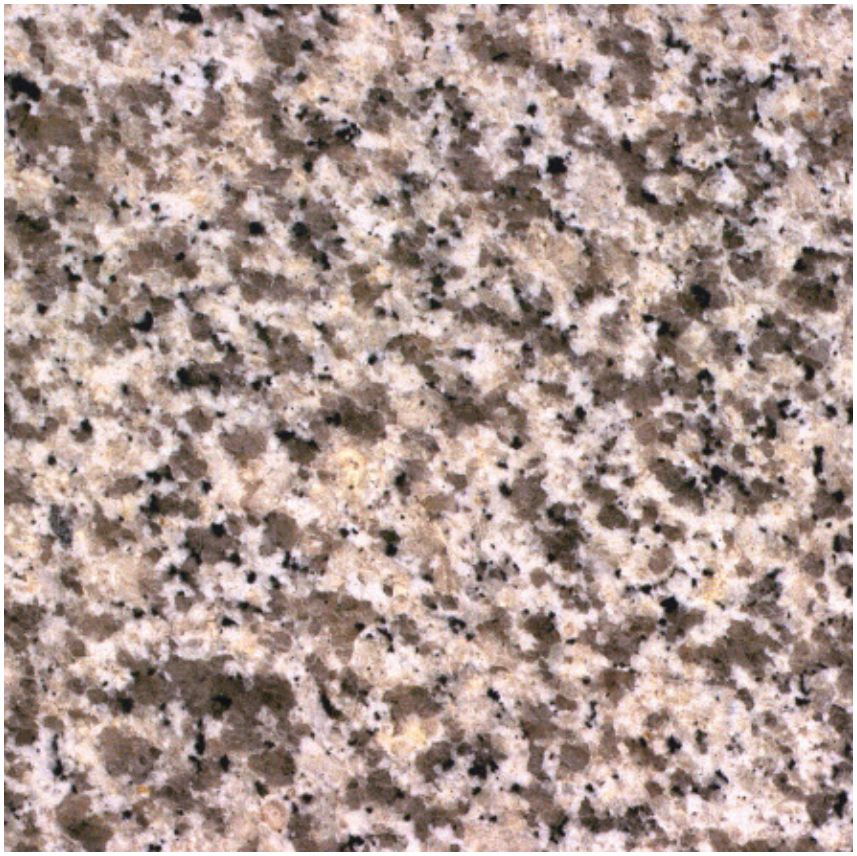
- MIP stands for *multum in parvo* = "*much in little*" (Williams 1983)

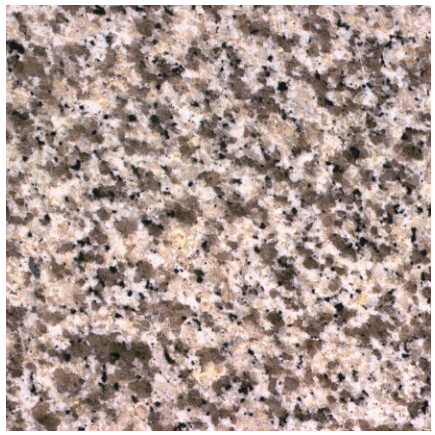
Before rendering

- Pre-compute and store down scaled versions of textures
  - Reduce resolution by factors of two successively
  - Use high quality filtering (averaging) scheme
- Increases memory cost by 1/3
  - 1/3 = ¼+1/16+1/64+…
- Width and height of texture need to be powers of two

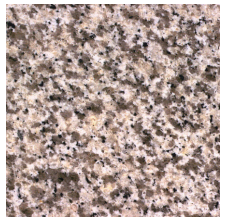
Mipmaps

- Example: resolutions 512x512, 256x256, 128x128, 64x64, 32x32 pixels

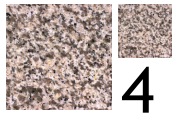




Level 1



2



3

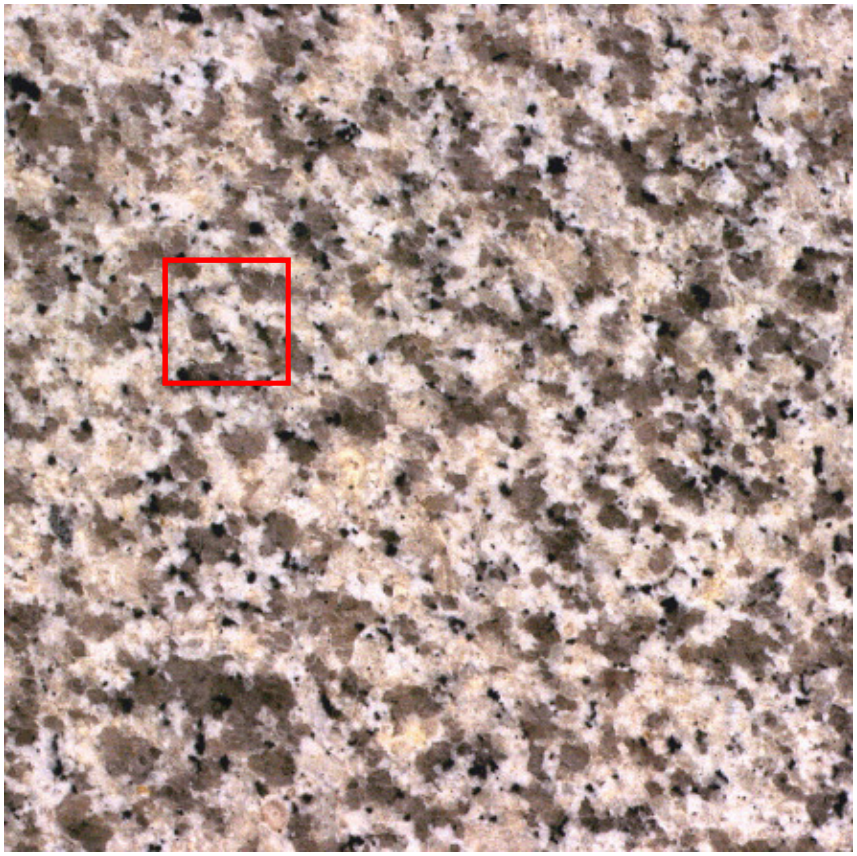
4

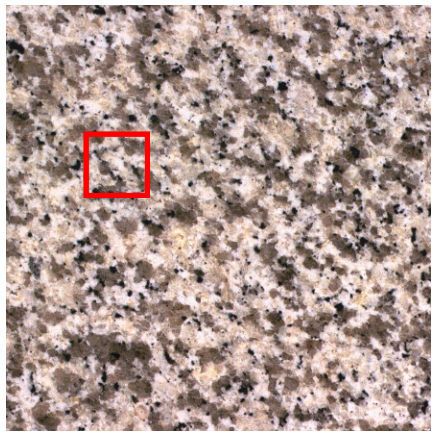
“multum in parvo”

Level 0

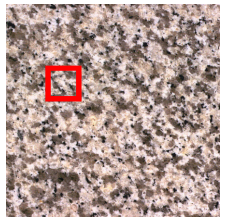
Mipmaps

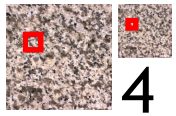
- One texel in level 4 is the average of 4^4=256 texels in level 0





Level 1





4

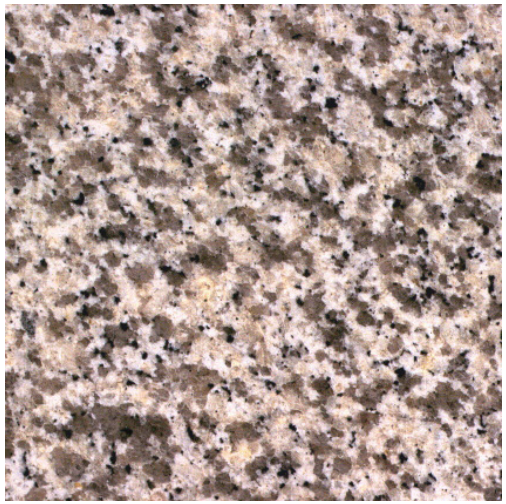
3

2

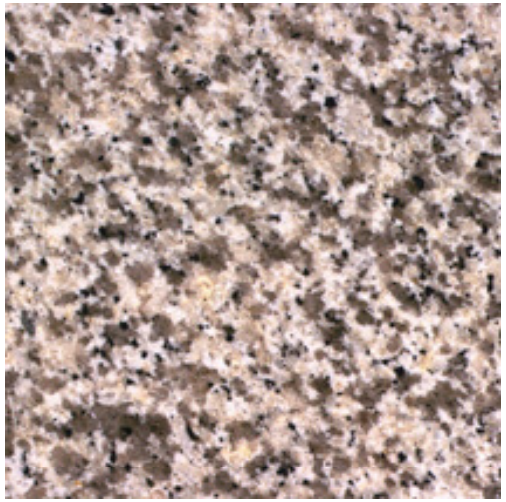
“multum in parvo”

Level 0

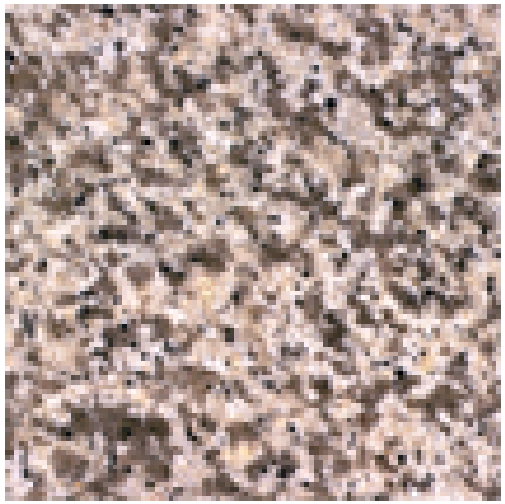
Mipmaps



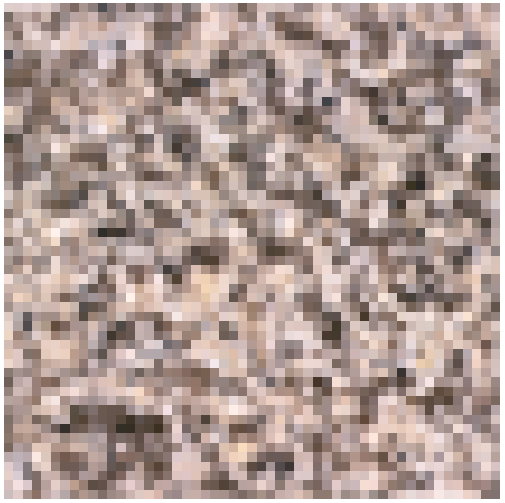
Level 0



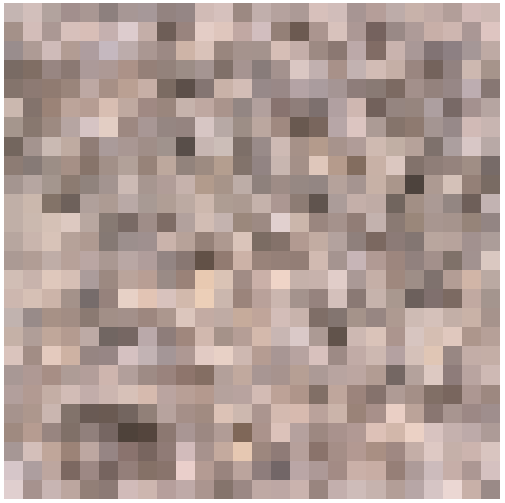
Level 1



Level 2



Level 3



Level 4

Rendering With Mipmaps

- "Mipmapping"
- Interpolate texture coordinates of each pixel as without mipmapping
- Compute approximate size of pixel in texture space
- Look up color in nearest mipmap
  - E.g., if pixel corresponds to 10x10 texels use mipmap level 3
  - Use nearest neighbor or bilinear interpolation as before

Mipmapping

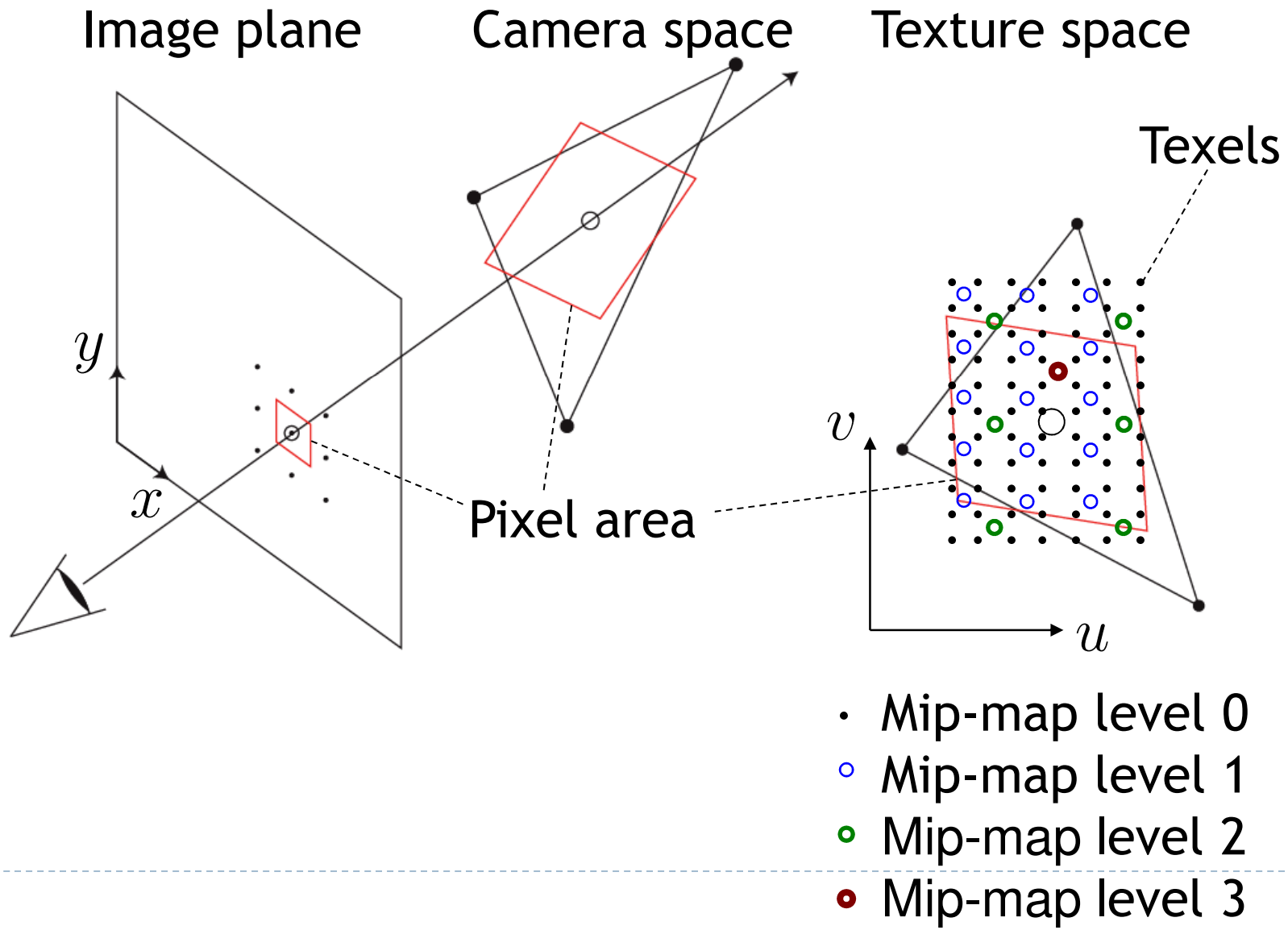
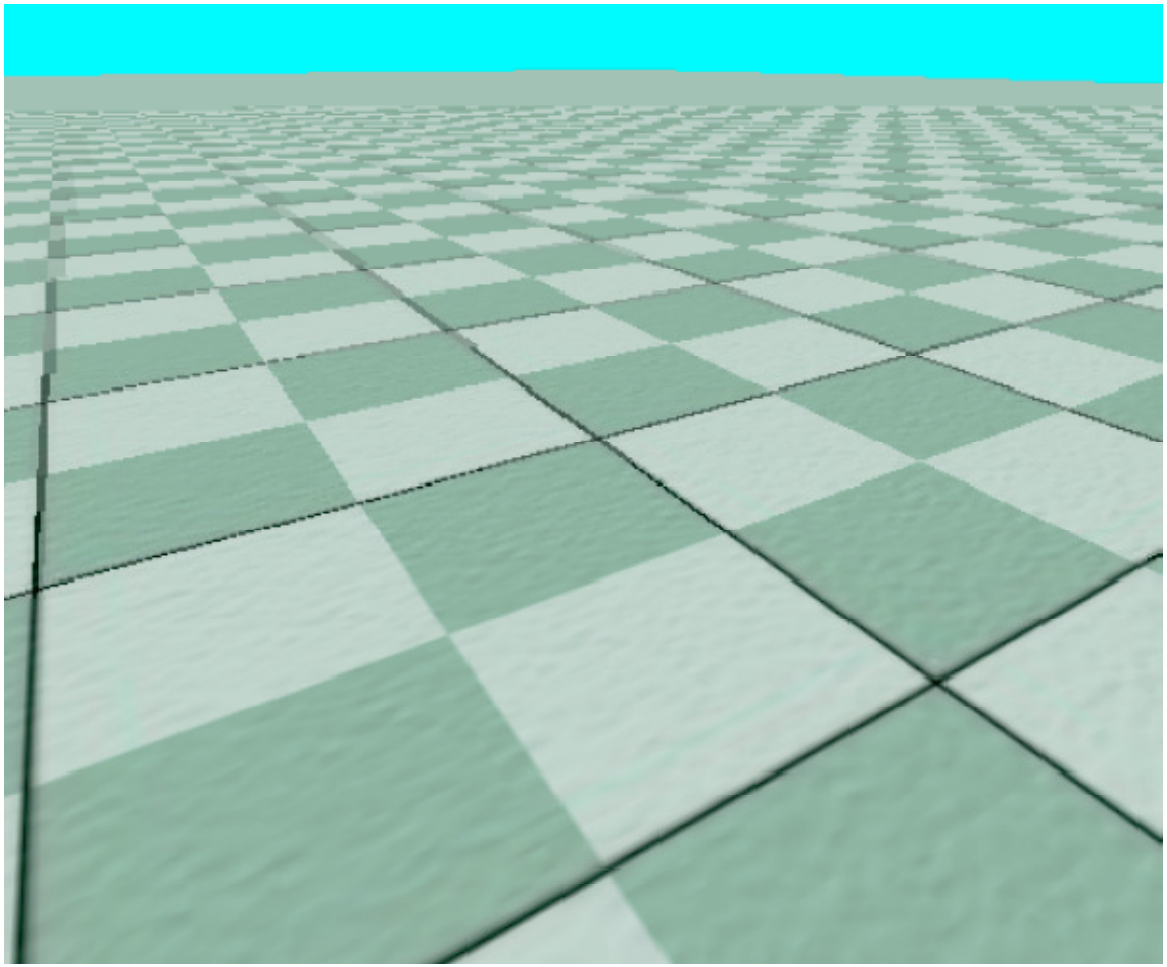


Image plane
Camera space
Texture space
Texels
y
x
Pixel area
v
u
Mip-map level 0
Mip-map level 1
Mip-map level 2
Mip-map level 3

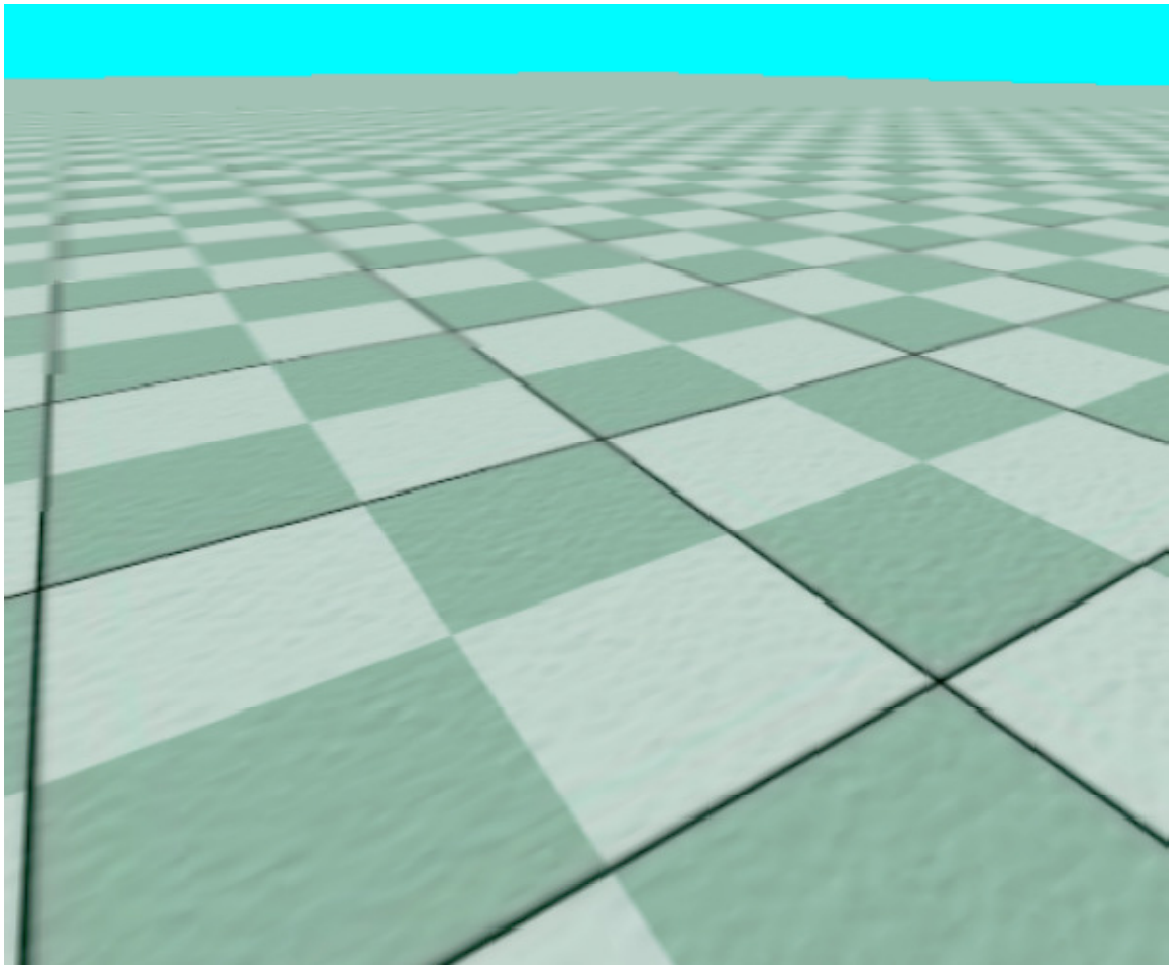
Nearest Mipmap, Nearest Neighbor

- Visible transition between mipmap levels



Nearest Mipmap, Bilinear

- Visible transition between mipmap levels

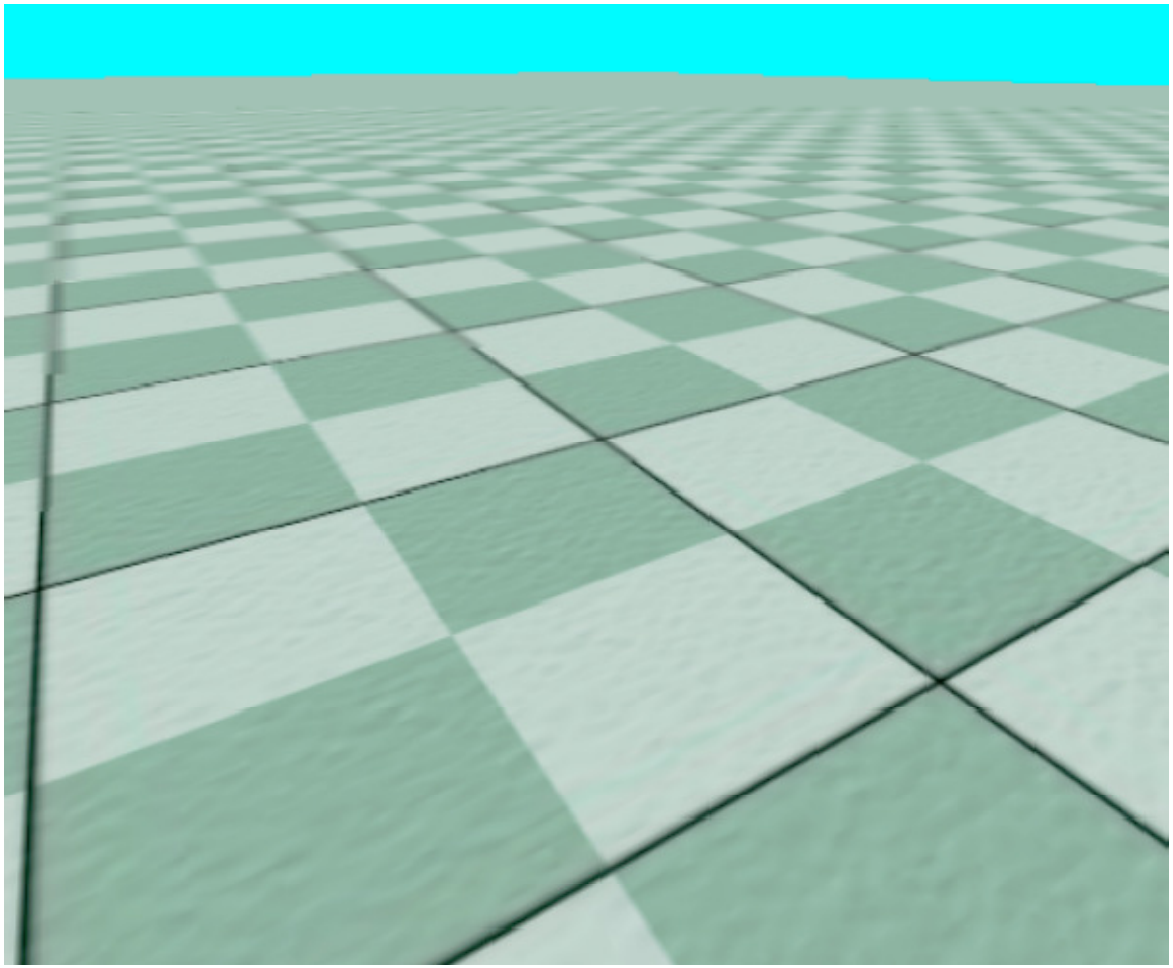


Trilinear Mipmapping

- Use two nearest mipmap levels
  - E.g., if pixel corresponds to 10x10 texels, use mipmap levels 3 (8x8) and 4 (16x16)
- Perform bilinear interpolation in both mip-maps
- Linearly interpolate between the results
- Requires access to 8 texels for each pixel
- Standard method, supported by hardware with no performance penalty

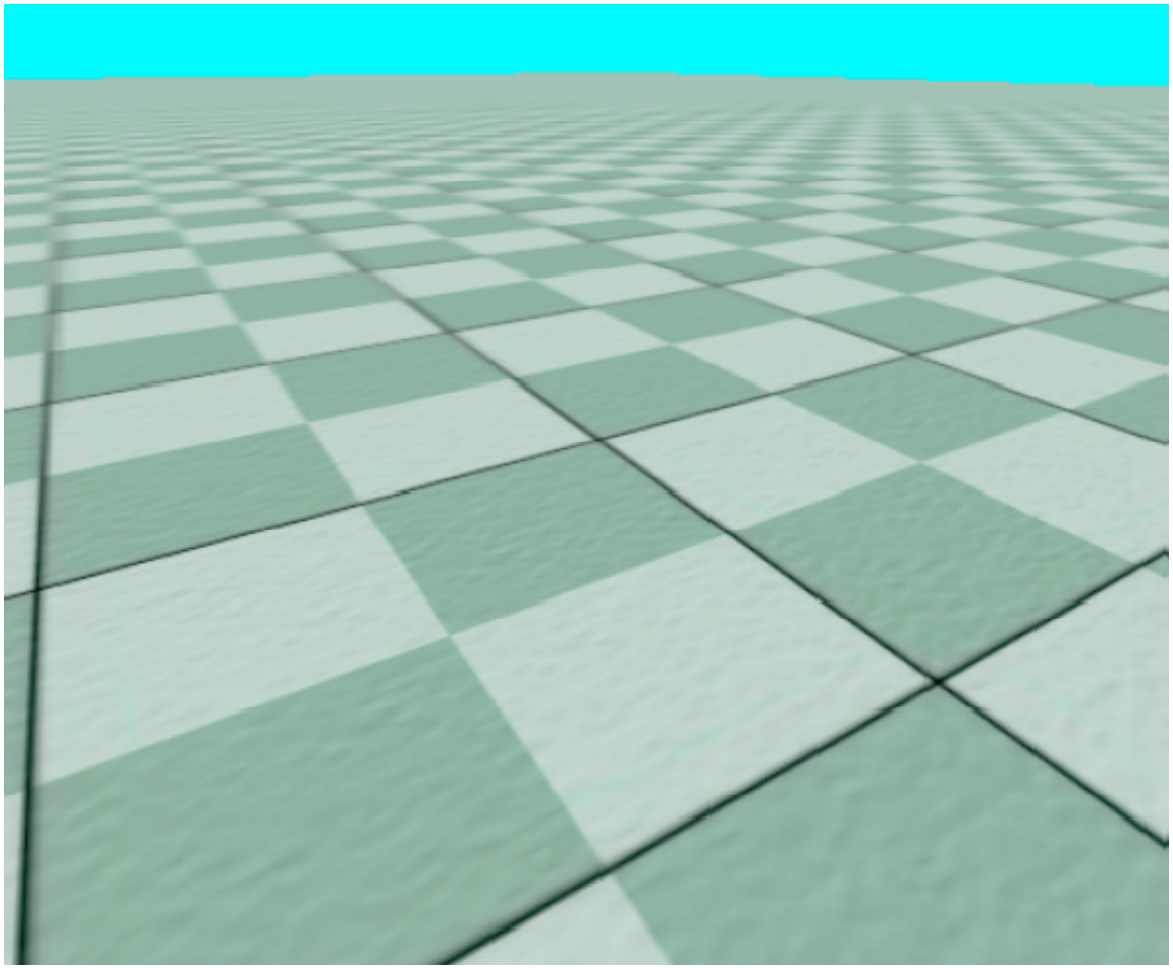
Nearest Mipmap, Bilinear

- Visible transition between mipmap levels



Trilinear Mipmapping

- Smooth transition between mipmap levels



Next Lecture

- Thursday:
  - Midterm Exam
- Next Tuesday:
  - Advanced Texture Mapping Techniques