418382 สภาพแวดล้อมการทำงานคอมพิวเตอร์กราฟิกส์
การบรรยายครั้งที่ 5

ประมุข ขันเงิน

pramook@gmail.com

15-462 Computer Graphics I

Lecture 8

Shading in OpenGL

Polygonal Shading
Light Source in OpenGL
Material Properties in OpenGL
Normal Vectors in OpenGL
Approximating a Sphere

[Angel 6.5-6.9]

February 6, 2003

Frank Pfenning
Carnegie Mellon University

http://www.cs.cmu.edu/~fp/courses/graphics/

Polygonal Shading

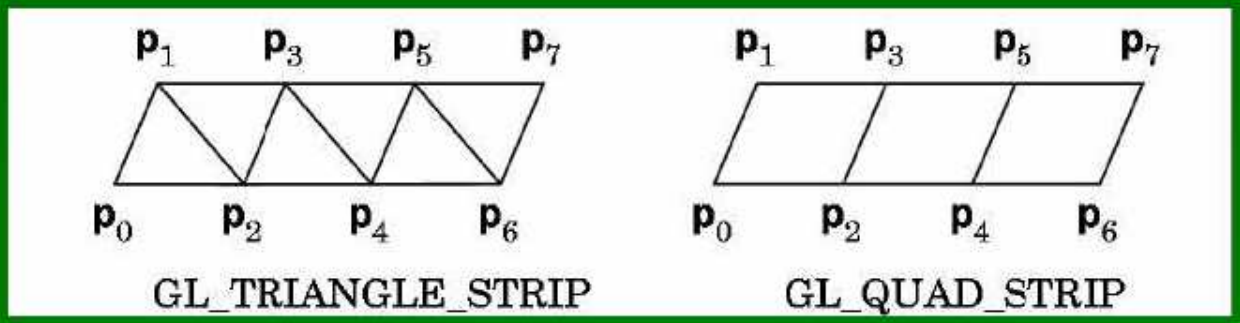
- Curved surfaces are approximated by polygons
- How do we shade?
  - Flat shading
  - Interpolative shading
  - Gouraud shading
  - Phong shading (different from Phong illumination)
- Two questions:
  - How do we determine normals at vertices?
  - How do we calculate shading at interior points?

Flat Shading

- Normal: given explicitly before vertex

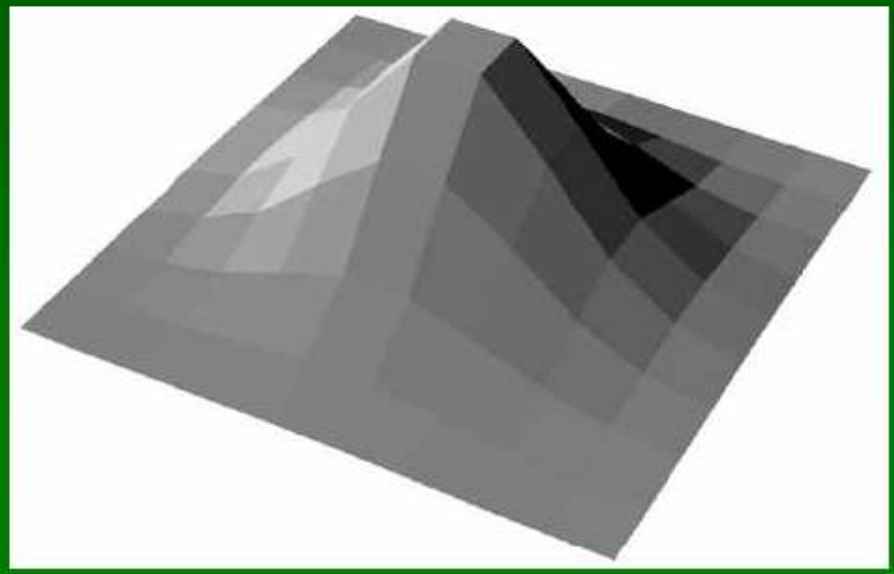
```
glNormal3f(nx, ny, nz);
glVertex3f(x, y, z);
```

- Shading constant across polygon
- Single polygon: first vertex
- Triangle strip:Vertex n+2 for triangle n



Flat Shading Assessment

- Inexpensive to compute
- Appropriate for objects with flat faces
- Less pleasant for smooth surfaces



Interpolative Shading

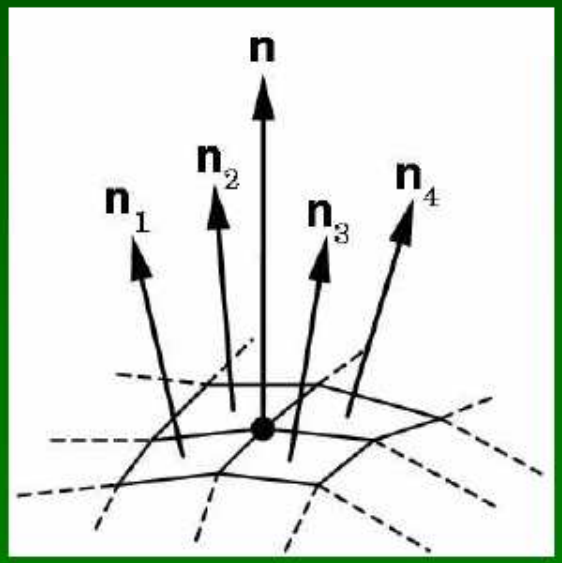
- Enable with glShadeModel(GL_SMOOTH);
- Calculate color at each vertex
- Interpolate color in interior
- Compute during scan conversion (rasterization)
- Much better image (see Assignment 1)
- More expensive to calculate

Gouraud Shading

- Special case of interpolative shading
- How do we calculate vertex normals?
- Gouraud: average all adjacent face normals

$$\mathbf{n} = \frac{\mathbf{n}_1 + \mathbf{n}_2 + \mathbf{n}_3 + \mathbf{n}_4}{|\mathbf{n}_1 + \mathbf{n}_2 + \mathbf{n}_3 + \mathbf{n}_4|}$$

- Requires knowledge about which faces share a vertex

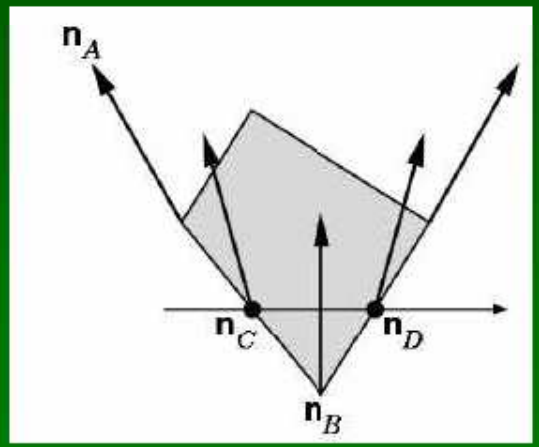


Data Structures for Gouraud Shading

- Sometimes vertex normals can be computed directly (e.g. height field with uniform mesh)
- More generally, need data structure for mesh
- Key: which polygons meet at each vertex

Phong Shading

- Interpolate normals rather than colors
- Significantly more expensive
- Mostly done off-line (not supported in OpenGL)



Phong Shading Results

Michael Gold, Nvidia

Single pass Phong Lighting Gouraud Shading	Two pass Phong Lighting, Gouraud Shading	Two pass Phong Lighting, Phong Shading

Polygonal Shading Summary

- Gouraud shading
  - Set vertex normals
  - Calculate colors at vertices
  - Interpolate colors across polygon
- Must calculate vertex normals!
- Must normalize vertex normals to unit length!

Outline

- Polygonal Shading
- Light Sources in OpenGL
- Material Properties in OpenGL
- Normal Vectors in OpenGL
- Example: Approximating a Sphere

Enabling Lighting and Lights

- Lighting in general must be enabled

```
glEnable(GL_LIGHTING);
```

- Each individual light must be enabled

```
glEnable(GL_LIGHT0);
```

- OpenGL supports at least 8 light sources

Global Ambient Light

- Set ambient intensity for entire scene

```
GLfloat al[] = {0.2, 0.2, 0.2, 1.0};
glLightModelfv(GL_LIGHT_MODEL_AMBIENT, al);
```

- The above is default
- Also: local vs infinite viewer

```
glLightModeli(GL_LIGHT_MODEL_LOCAL_VIEWER,
              GL_TRUE);
```

- More expensive, but sometimes more accurate

Defining a Light Source

- Use vectors {r, g, b, a} for light properties
- Beware: light source will be transformed!

```
GLfloat light_ambient[] = {0.2, 0.2, 0.2, 1.0};
GLfloat light_diffuse[] = {1.0, 1.0, 1.0, 1.0};
GLfloat light_specular[] = {1.0, 1.0, 1.0, 1.0};
GLfloat light_position[] = {-1.0, 1.0, -1.0, 0.0};
glLightfv(GL_LIGHT0, GL_AMBIENT, light_ambient);
glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse);
glLightfv(GL_LIGHT0, GL_SPECULAR, light_specular);
glLightfv(GL_LIGHT0, GL_POSITION, light_position);
```

Point Source vs Directional Source

- Directional light given by “position” vector

```
GLfloat light_position[] = {-1.0, 1.0, -1.0, 0.0};
glLightfv(GL_LIGHT0, GL_POSITION, light_position);
```

- Point source given by “position” point

```
GLfloat light_position[] = {-1.0, 1.0, -1.0, 1.0};
glLightfv(GL_LIGHT0, GL_POSITION, light_position);
```

Spotlights

- Create point source as before
- Specify additional properties to create spotlight

```
GLfloat sd[] = {-1.0, -1.0, 0.0};
glLightfv(GL_LIGHT0, GL_SPOT_DIRECTION, sd);
glLightf(GL_LIGHT0, GL_SPOT_CUTOFF, 45.0);
glLightf(GL_LIGHT0, GL_SPOT_EXPONENT, 2.0);
```

[Demo: Lighting Position Tutor]

Example Lighting Properties

```
GLfloat light_ambient[]={0.2, 0.2, 0.2, 1.0};
GLfloat light_diffuse[]={1.0, 1.0, 1.0, 1.0};
GLfloat light_specular[]={0.0, 0.0, 0.0, 1.0};

glLightfv(GL_LIGHT0, GL_AMBIENT, light_ambient);
glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse);
glLightfv(GL_LIGHT0, GL_SPECULAR, light_specular);
```

Outline

- Polygonal Shading
- Light Sources in OpenGL
- Material Properties in OpenGL
- Normal Vectors in OpenGL
- Example: Approximating a Sphere

Defining Material Properties

- Material properties stay in effect
- Set both specular coefficients and shininess

```
GLfloat mat_d[] = {0.1, 0.5, 0.8, 1.0};
GLfloat mat_s[] = {1.0, 1.0, 1.0, 1.0};
GLfloat low_sh[] = {5.0};
glMaterialfv(GL_FRONT, GL_AMBIENT, mat_d);
glMaterialfv(GL_FRONT, GL_SPECULAR, mat_s);
glMaterialfv(GL_FRONT, GL_SHININESS, low_sh);
```

- Diffuse component is analogous

[Demo: Light material Tutor]

Color Material Mode (Answer)

- Can shortcut material properties using glColor
- Must be explicitly enabled and disabled

```
glEnable(GL_COLOR_MATERIAL);
/* affect front face, diffuse reflection properties */
glColorMaterial(GL_FRONT, GL_DIFFUSE);
glColor3f(0.0, 0.0, 0.8);
/* draw some objects here in blue */
glColor3f(1.0, 0.0, 0.0);
/* draw some objects here in red */
glDisable(GL_COLOR_MATERIAL);
```

Example Material Properties

```
GLfloat mat_specular[]={0.0, 0.0, 0.0, 1.0};
GLfloat mat_diffuse[]={0.8, 0.6, 0.4, 1.0};
GLfloat mat_ambient[]={0.8, 0.6, 0.4, 1.0};
GLfloat mat_shininess={20.0};
glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient);
glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
glMaterialf(GL_FRONT, GL_SHININESS, mat_shininess);

glShadeModel(GL_SMOOTH); /*enable smooth shading */
glEnable(GL_LIGHTING); /* enable lighting */
glEnable(GL_LIGHT0);  /* enable light 0 */
```

Outline

- Polygonal Shading
- Light Sources in OpenGL
- Material Properties in OpenGL
- Normal Vectors in OpenGL
- Example: Approximating a Sphere

Defining and Maintaining Normals

- Define unit normal before each vertex

```
glNormal3f(nx, ny, nz);
glVertex3f(x, y, z);
```

- Length changes under some transformations
- Ask OpenGL to re-normalize (all tfms)

```
glEnable(GL_NORMALIZE);
```

- Ask OpenGL to re-scale normal

```
glEnable(GL_RESCALE_NORMAL);
```

- Works for uniform scaling (and rotate, translate)

Example: Icosahedron

- Define the vertices

```
#define X .525731112119133606
#define Z .850650808352039932

static GLfloat vdata[12][3] = {
  {-X, 0.0, Z}, {X, 0.0, Z}, {-X, 0.0, -Z}, {X, 0.0, -Z},
  {0.0, Z, X}, {0.0, Z, -X}, {0.0, -Z, X}, {0.0, -Z, -X},
  {Z, X, 0.0}, {-Z, X, 0.0}, {Z, -X, 0.0}, {-Z, -X, 0.0}
};
```

- For simplicity, avoid the use of vertex arrays

Defining the Faces

- Index into vertex data array

```
static GLuint tindices[20][3] = {
  {1,4,0}, {4,9,0}, {4,9,5}, {8,5,4}, {1,8,4},
  {1,10,8}, {10,3,8}, {8,3,5}, {3,2,5}, {3,7,2},
  {3,10,7}, {10,6,7}, {6,11,7}, {6,0,11}, {6,1,0},
  {10,1,6}, {11,0,9}, {2,11,9}, {5,2,9}, {11,2,7}
};
```

- Be careful about orientation!

Drawing the Icosahedron

- Normal vector calculation next

```
glBegin(GL_TRIANGLES);
for (i = 0; i < 20; i++) {
  icoNormVec(i);
  glVertex3fv(&vdata[tindices[i][0]] [0]);
  glVertex3fv(&vdata[tindices[i][1]] [0]);
  glVertex3fv(&vdata[tindices[i][2]] [0]);
}
glEnd();
```

- Should be encapsulated in display list

Calculating the Normal Vectors

- Normalized cross product of any two sides

```
GLfloat d1[3], d2[3], n[3];

void icoNormVec (int i) {
  for (k = 0; k < 3; k++) {
    d1[k] = vdata[tindices[i][0]] [k] – vdata[tindices[i][1]] [k];
    d2[k] = vdata[tindices[i][1]] [k] – vdata[tindices[i][2]] [k];
  }
normCrossProd(d1, d2, n);
glNormal3fv(n);
}
```

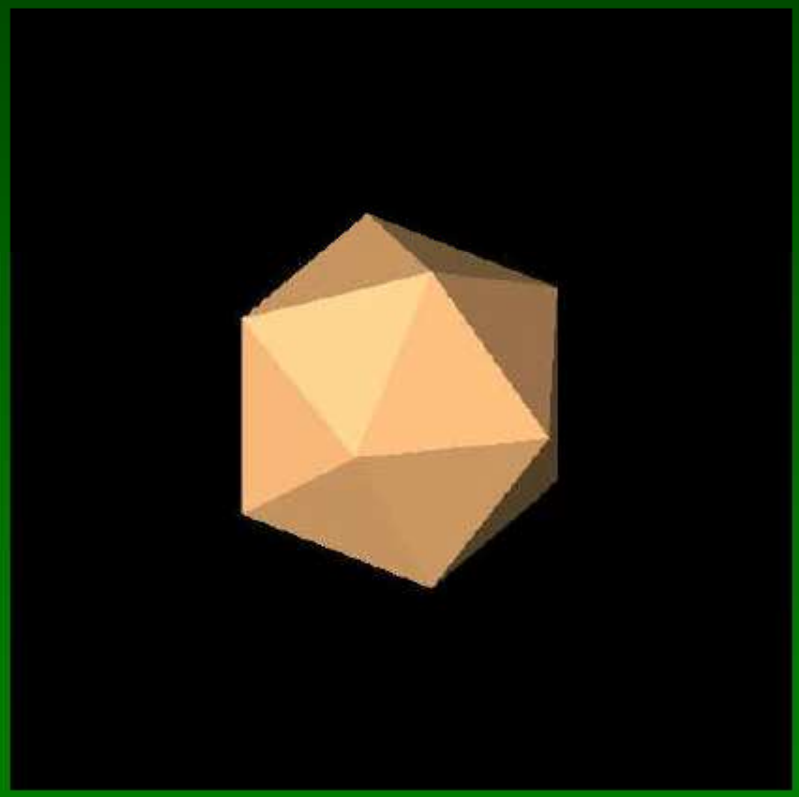
The Normalized Cross Product

- Omit zero-check for brevity

```
void normalize(float v[3]) {
  GLfloat d = sqrt(v[0]*v[0] + v[1]*v[1] + v[2]*v[2]);
  v[0] /= d; v[1] /= d; v[2] /= d;
}

void normCrossProd(float u[3], float v[3], float out[3]) {
  out[0] = u[1]*v[2] – u[2]*v[1];
  out[1] = u[2]*v[0] – u[0]*v[2];
  out[2] = u[0]*v[1] – u[1]*v[0];
  normalize(out);
}
```

The Icosahedron

- Using simple lighting setup



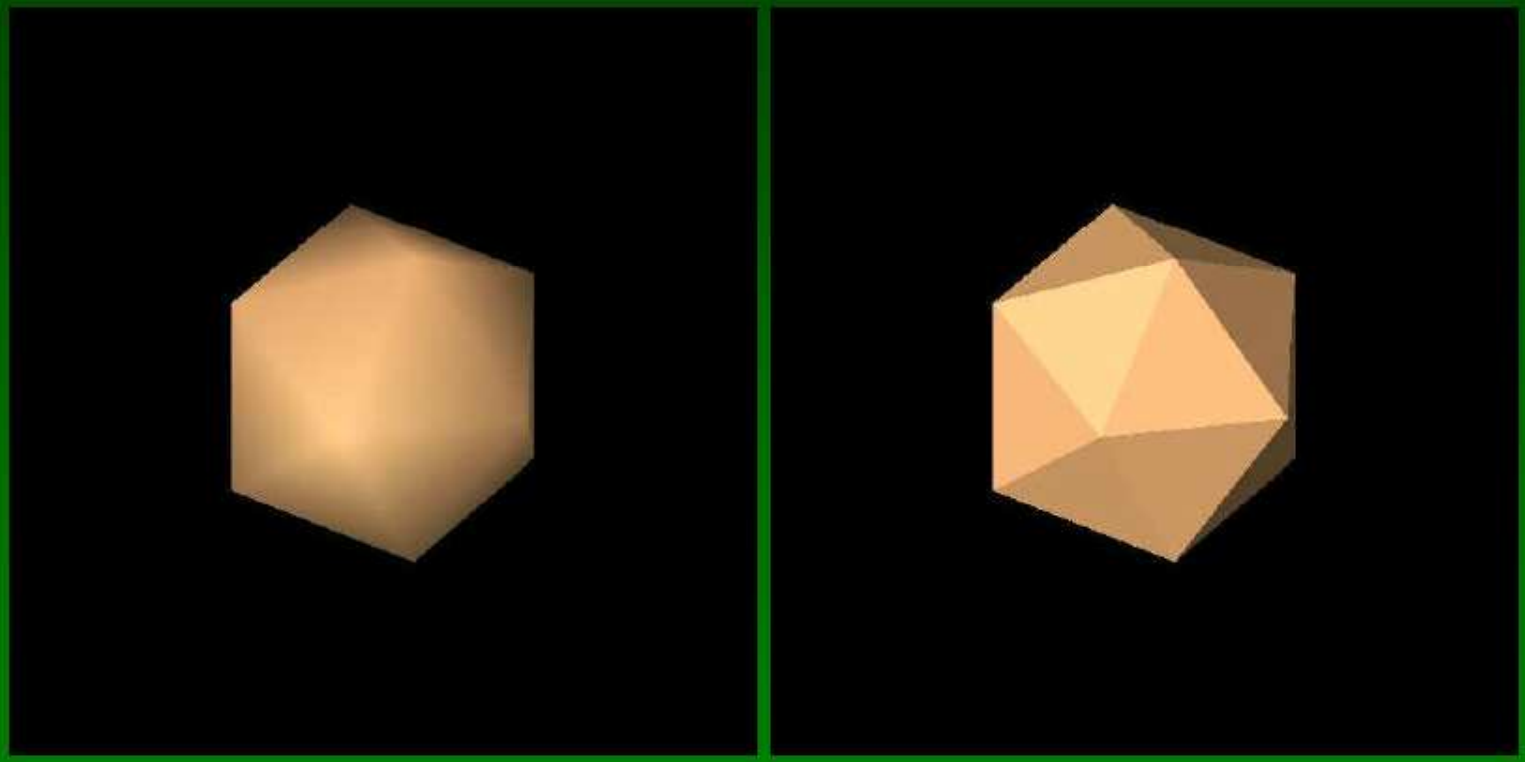
Sphere Normals

- Set up instead to use normals of sphere
- Unit sphere normal is exactly sphere point

```
glBegin(GL_TRIANGLES);
 for (i = 0; i < 20; i++) {
   glNormal3fv(&vdata[tindices[i][0]][0]);
   glVertex3fv(&vdata[tindices[i][0]][0]);
   glNormal3fv(&vdata[tindices[i][1]][0]);
   glVertex3fv(&vdata[tindices[i][1]][0]);
   glNormal3fv(&vdata[tindices[i][2]][0]);
   glVertex3fv(&vdata[tindices[i][2]][0]);
 }
 glEnd();
```

Icosahedron with Sphere Normals

- Interpolation vs flat shading effect



TEXTURE MAPPING

Texture Mapping

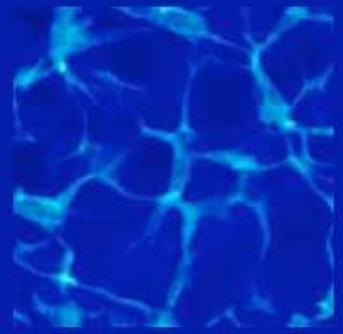
- A way of adding surface details
- Two ways can achieve the goal:
  - Model the surface with more polygons
    - » Slows down rendering speed
    - » Hard to model fine features

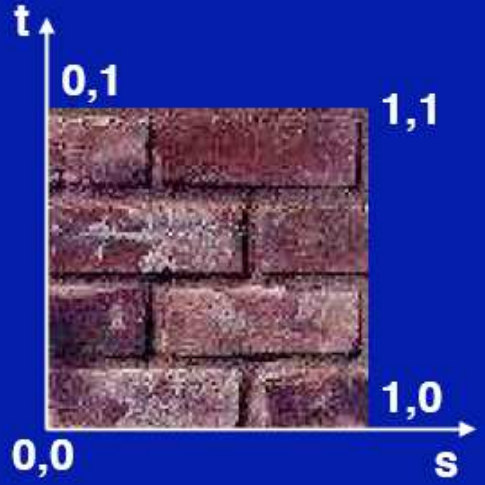
Texture Mapping

- A way of adding surface details
- Two ways can achieve the goal:
  - Model the surface with more polygons
    - » Slows down rendering speed
    - » Hard to model fine features
  - Map a texture to the surface
    - » This lecture
    - » Image complexity does not affect complexity of processing

The texture

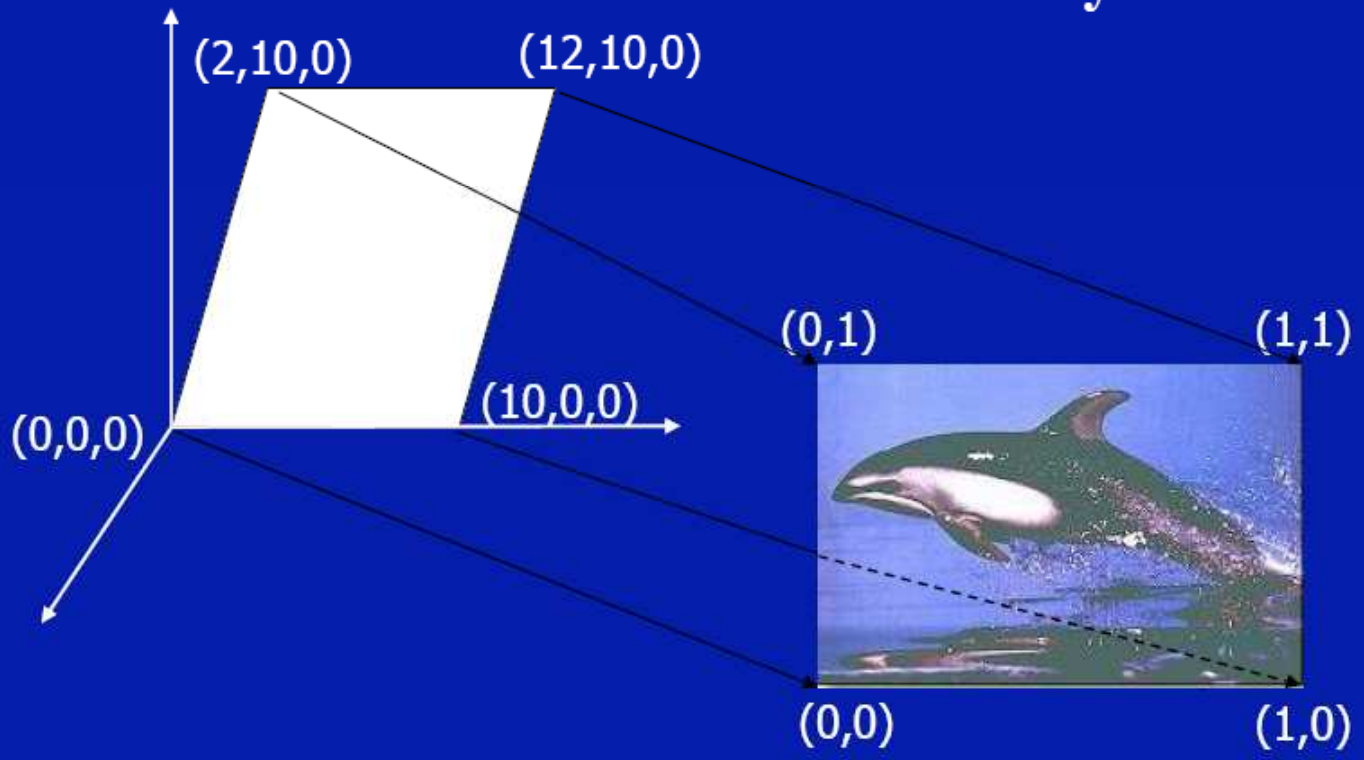
- Texture is a bitmap image
- 2D array: texture[height][width][4]
- Pixels of the texture called *texels*
- Texel coordinates (s,t) scaled to [0,1] range





Map textures to surfaces

The polygon can have arbitrary size and shape



The drawing itself

- Use GLTexCoord2f(s,t) to specify texture coordinates
- Example:

```
glEnable(GL_TEXTURE_2D)
glBegin(GL_QUADS);
glTexCoord2f(0.0,0.0);  glVertex3f(0.0,0.0,0.0);
glTexCoord2f(0.0,1.0);  glVertex3f(2.0,10.0,0.0);
glTexCoord2f(1.0,0.0);  glVertex3f(10.0,0.0,0.0);
glTexCoord2f(1.0,1.0);  glVertex3f(12.0,10.0,0.0);
glEnd();
glDisable(GL_TEXTURE_2D)
```

- State machine: Texture coordinates remain valid until you change them or exit texture mode via

  glDisable (GL_TEXTURE_2D)

Color blending

- Final pixel color = f (texture color, object color)
- How to determine the color of the final pixel?
  - GL_REPLACE – use texture color to replace object color
  - GL_BLEND – linear combination of texture and object color
  - GL_MODULATE – multiply texture and object color
- Example:
  - glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE);

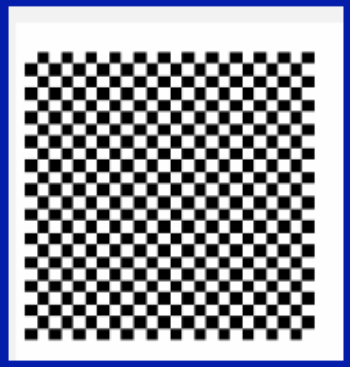
What happens if texture coordinates outside [0,1] ?

- Two choices:
  - Repeat pattern (GL_REPEAT)
  - Clamp to maximum/minimum value (GL_CLAMP)

- Example:
  - glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP)
  - glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP)

What happens if texture coordinates outside [0,1] ?

clamp

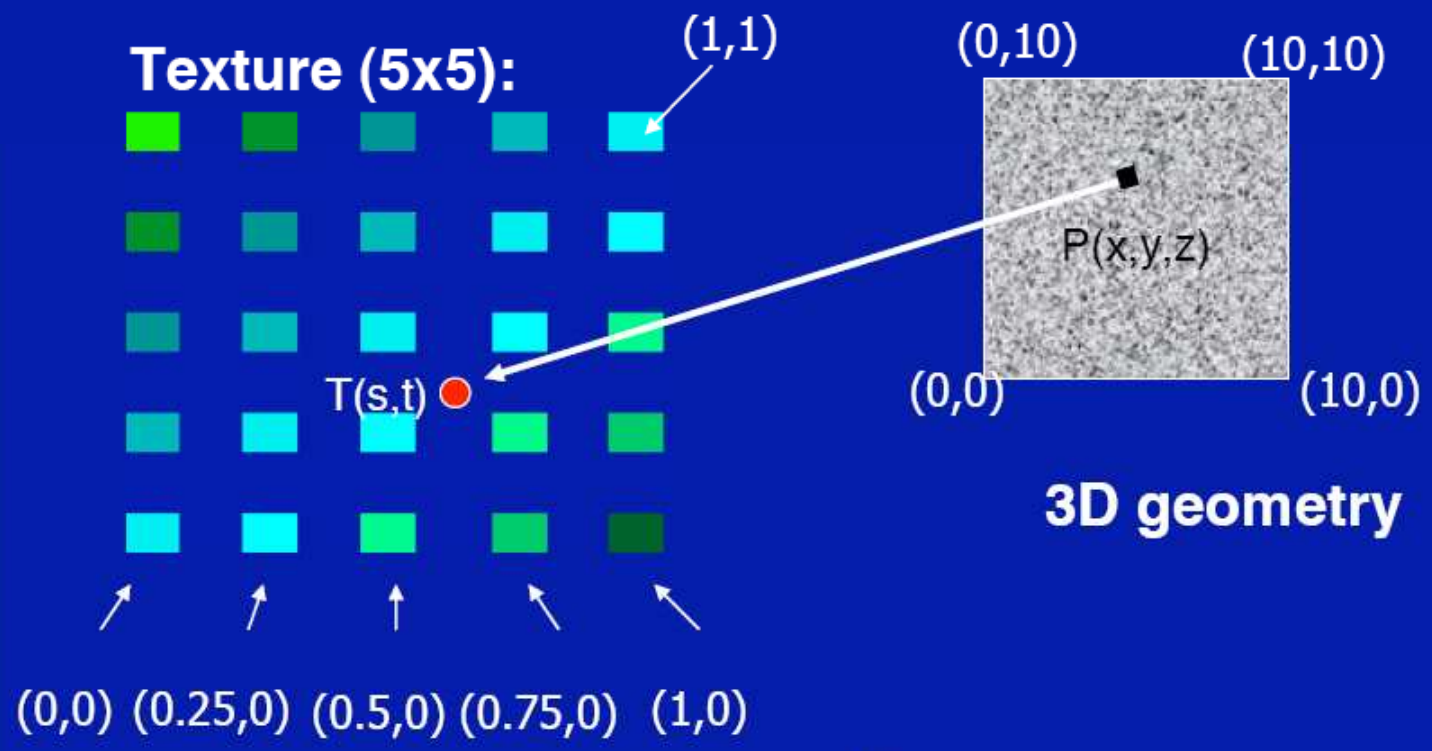


repeat

```
glTexCoord2f(0.0, 0.0);  glVertex3f(0.0,  0.0,  0.0);
glTexCoord2f(0.0, 3.0);  glVertex3f(0.0, 10.0, 0.0);
glTexCoord2f(3.0, 0.0);  glVertex3f(10.0, 0.0, 0.0);
glTexCoord2f(3.0, 3.0);  glVertex3f(10.0, 10.0, 0.0);
```

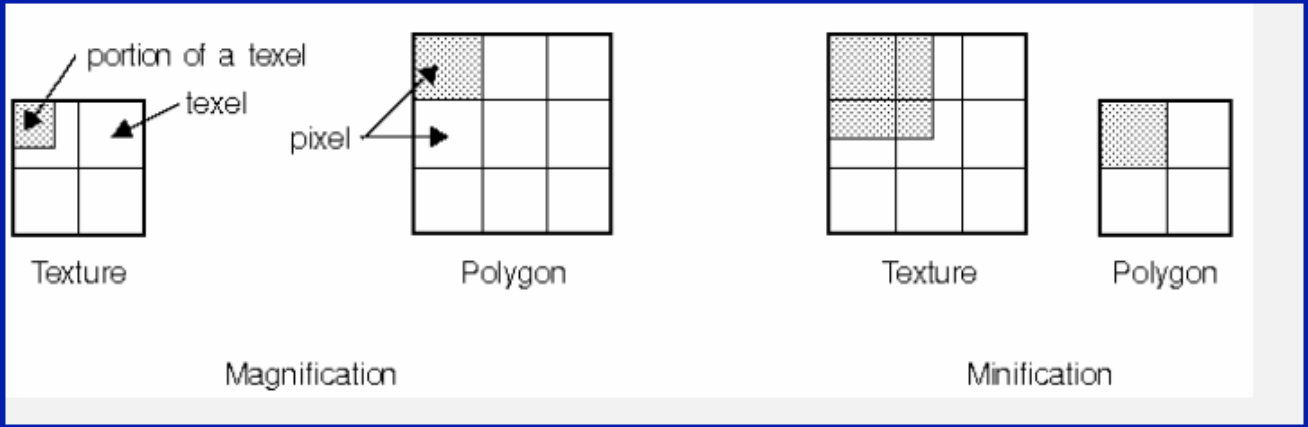
Texture Value Lookup

- For given texture coordinates (s,t), we can find a unique image value, corresponding to the texture image at that location



Interpolating colors

- **Some (s,t) coordinates not directly at pixel in the texture, but in between**



Interpolating colors

- **Solutions:**
  - **Nearest neighbor**
    - » **Use the nearest neighbor to determine color**
    - » **Faster, but worse quality**
    - » **glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);**
  - **Linear interpolation**
    - » **Incorporate colors of several neighbors to determine color**
    - » **Slower, better quality**
    - » **glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR)**

Textures

Texture Object

- An OpenGL data type that keeps textures resident in memory and provides identifiers to easily access them
- Provides efficiency gains over having to repeatedly load and reload a texture
- You can prioritize textures to keep in memory
- OpenGL uses least recently used (LRU) if no priority is assigned

Step 1 – create Texture Objects

```
glGenTextures(1, &texture[texture_num]);
```

- First argument tells GL how many Texture Objects to create
- Second argument is a pointer to the place where OpenGL will store the names (unsigned integers) of the Texture Objects it creates
  - texture[ ] is of type GLuint

Step 2 – Specify which texture object is about to be defined

Tell OpenGL that you are going to define the specifics of the Texture Object it created

- `glBindTexture(GL_TEXTURE_2D, texture[texture_num]);`
  - Textures can be 1D and 3D as well

Step 3 – Begin defining texture

glTexParameter()

- Sets various parameters that control how a texture is treated as it's applied to a fragment or stored in a texture object
- // scale linearly when image bigger than texture

```
glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_
FILTER,GL_LINEAR);
```

- // scale linearly when image smaller than texture

```
glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_
FILTER,GL_LINEAR);
```

Step 4 – Assign image data

- `glTexImage2D();`

GL_TEXTURE_2D	*(2D Texture)*
0	*(level of detail 0)*
3	*(3 components, RGB)*
image1->sizeX	*(size)*
image1->sizeY	*(size)*
0	*(no border pixel)*
GL_RGB	*(RGB color order)*
GL_UNSIGNED_BYTE	*(unsigned byte data)*
image1->data	*(pointer to the data))*

glTexImage2D – Arg 1

GLenum target

- GL_TEXTURE_{1|2|3}D
- GL_PROXY_TEXTURE_2D
  - Provides queries for texture resources
  - Proceed with hypothetical texture use (GL won't apply the texture)
  - After query, call GLGetTexLevelParamter to verify presence of required system components
  - Doesn't check possibility of multiple texture interference

glTexImage2D – Arg 2

GLint level

- Used for Level of Detail (LOD)
- LOD stores multiple versions of texture that can be used at runtime (set of sizes)
- Runtime algorithms select appropriate version of texture
  - Pixel size of polygon used to select best texture
  - Eliminates need for error-prone filtering algorithms

glTexImage2D – Arg 3

GLint internalFormat

- GL defines 38 symbolic constants that describe which of R, G, B, and A are used in internal representation of texels
- Provides control over things texture can do
  - High bit depth alpha blending
  - High bit depth intensity mapping
  - General purpose RGB
- GL doesn't guarantee all options are available on given hardware

glTexImage2D – Args 4-6

GLsizei width

GLsizei height

- Dimensions of texture image
  - Must be $2^m + 2b$ (b=0 or 1 depending on border)
  - min, 64 x 64

GLint border

- Width of border (1 or 0)
  - Border allows linear blending between overlapping textures
  - Useful when manually tiling textures

glTexImage2D – Args 7 & 8

GLenum format

- Describe how texture data is stored in input array
  - GL_RGB, GL_RGBA, GL_BLUE…

GLenum type

- Data size of array components
  - GL_SHORT, GL_BYTE, GL_INT…

glTexImage2D – Arg 9

*Const GLvoid *texels*

- Pointer to data describing texture map

Step 5 – Apply texture

Before defining geometry

- `glEnable(GL_TEXTURE_2D);`
- `glBindTexture(GL_TEXTURE_2D, texture[0]);`
- `glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE);`

glTexEnv()

First argument to function is always GL_TEXTURE_ENV

GL_TEXTURE_ENV_MODE	*GL_DECAL (alpha blends texture with poly color)*
	GL_REPLACE (straight up replacement)
	GL_MODULATE (texture application is a function of poly lighting)
	GL_BLEND (texture controls blending with another color)
If GL_BLEND selected, second call to glTexEnv() must specify GL_TEXTURE_ENV_COLOR	*4-float array for R,G,B,A blend*

Enable/disable texture mode

- Can do in init() or successively in display()
- glEnable(GL_TEXTURE_2D)
- glDisable(GL_TEXTURE_2D)

- Successively enable/disable texture mode to switch between drawing textured/non-textured polygons
- Changing textures:
  - Only one texture active at any given time
  - make another call to glTexImage2D to make another pattern active

The drawing itself

- Use GLTexCoord2f(s,t) to specify texture coordinates
- State machine: Texture coordinates remain valid until you change them or exit texture mode via

  glDisable (GL_TEXTURE_2D)
- Example:

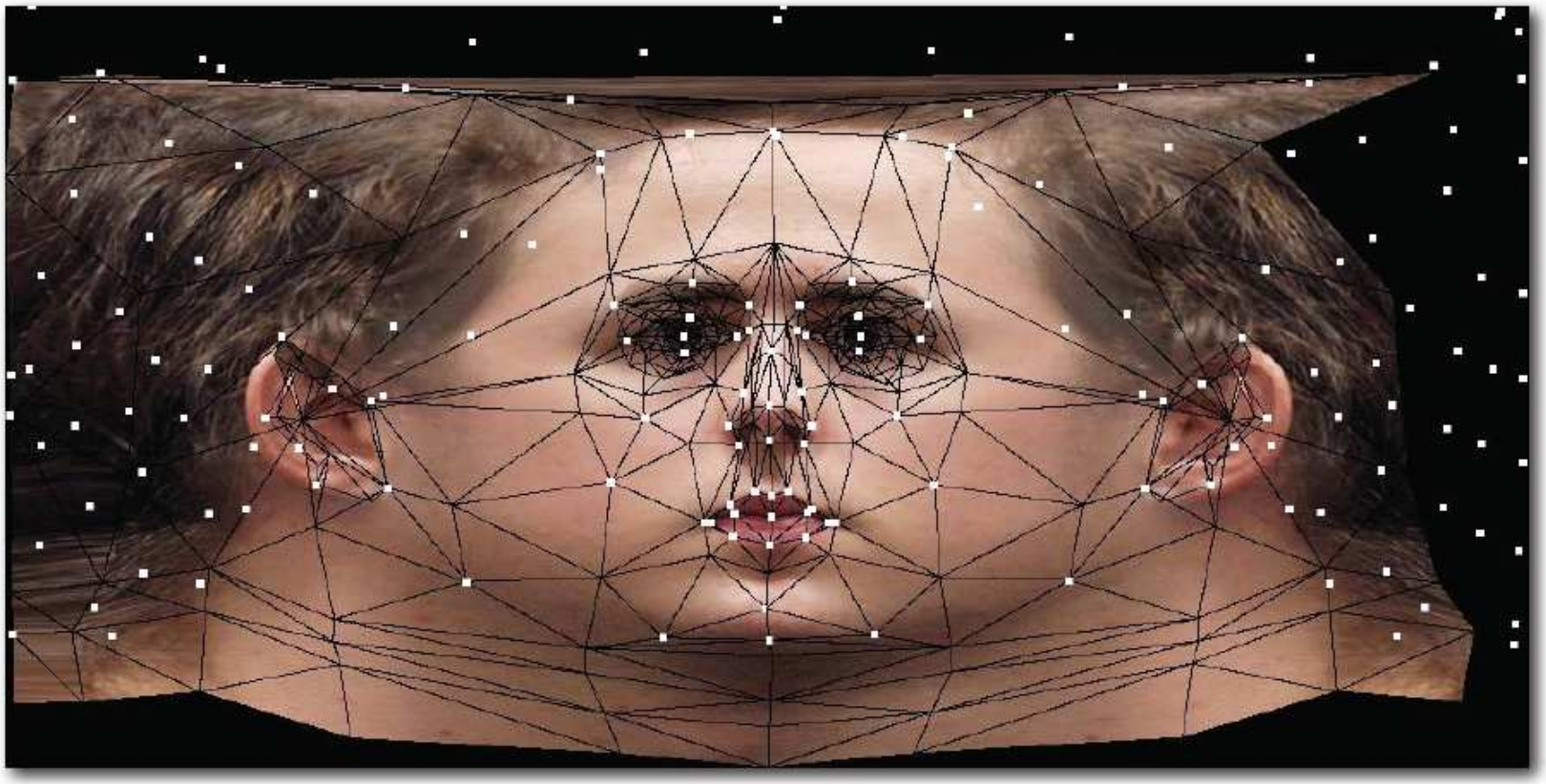
```
glEnable(GL_TEXTURE_2D)
glBegin(GL_QUADS);
glTexCoord2f(0.0,0.0); glVertex3f(-2.0,-1.0,0.0);
glTexCoord2f(0.0,1.0); glVertex3f(-2.0,1.0,0.0);
glTexCoord2f(1.0,0.0); glVertex3f(0.0,1.0,0.0);
glTexCoord2f(1.0,1.0); glVertex3f(0.0,-1.0,0.0);
…
glEnd();
glDisable(GL_TEXTURE_2D)
```

Everything together

```
void init(void):
{
...
put image into 2D memory array; // can use libpicio library

// specify texture parameters
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT); // repeat pattern in s
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT); // repeat pattern in t

// use nearest neighbor for both minification and magnification
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);

// make the pattern at location pointerToImage the active pattern
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, 256, 256, 0,
             GL_RGBA, GL_UNSIGNED_BYTE, pointerToImage)

...
}
```

Everything together (contd.)

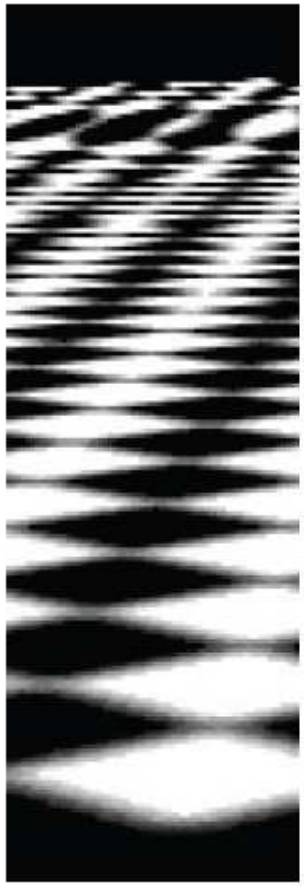
```
void display(void):
{
...
// no blending, use texture color directly
glTexEnvf(GL_TEXTURE_ENV,GL_TEXTURE_ENV_MODE, GL_REPLACE);
// turn on texture mode
glEnable(GL_TEXTURE_2D);

glBegin(GL_QUADS); // draw a quad
glTexCoord2f(0.0,0.0); glVertex3f(-2.0,-1.0,0.0);
glTexCoord2f(0.0,1.0); glVertex3f(-2.0,1.0,0.0);
glTexCoord2f(1.0,0.0); glVertex3f(0.0,1.0,0.0);
glTexCoord2f(1.0,1.0); glVertex3f(0.0,-1.0,0.0);
...
glEnd();

// turn off texture mode
glDisable(GL_TEXTURE_2D);

// draw some non-texture mapped objects
...
// switch back to texture mode, etc.
...
}
```



MIPMAPPING

Sampling Texture Maps

- When texture mapping it is rare that the screen-space pixel sampling density matches the sampling density of the texture
- Typically one of two things can occur:
  - Minification
  - Magnification

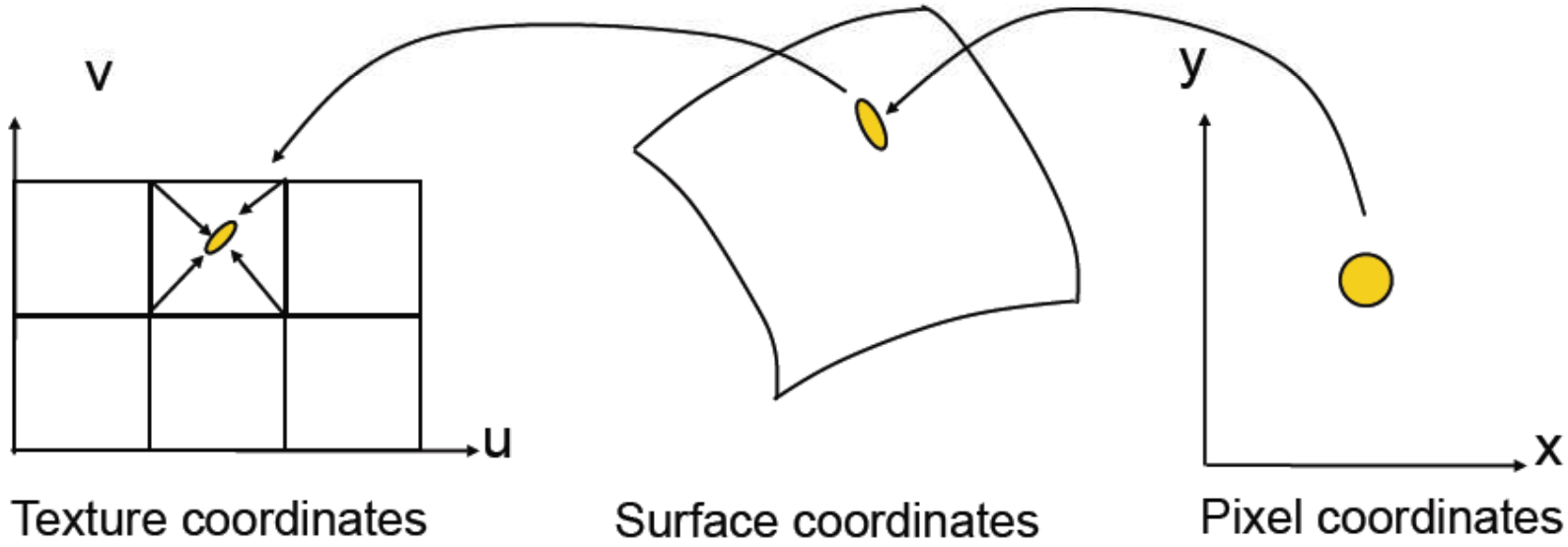


Minification

Magnification

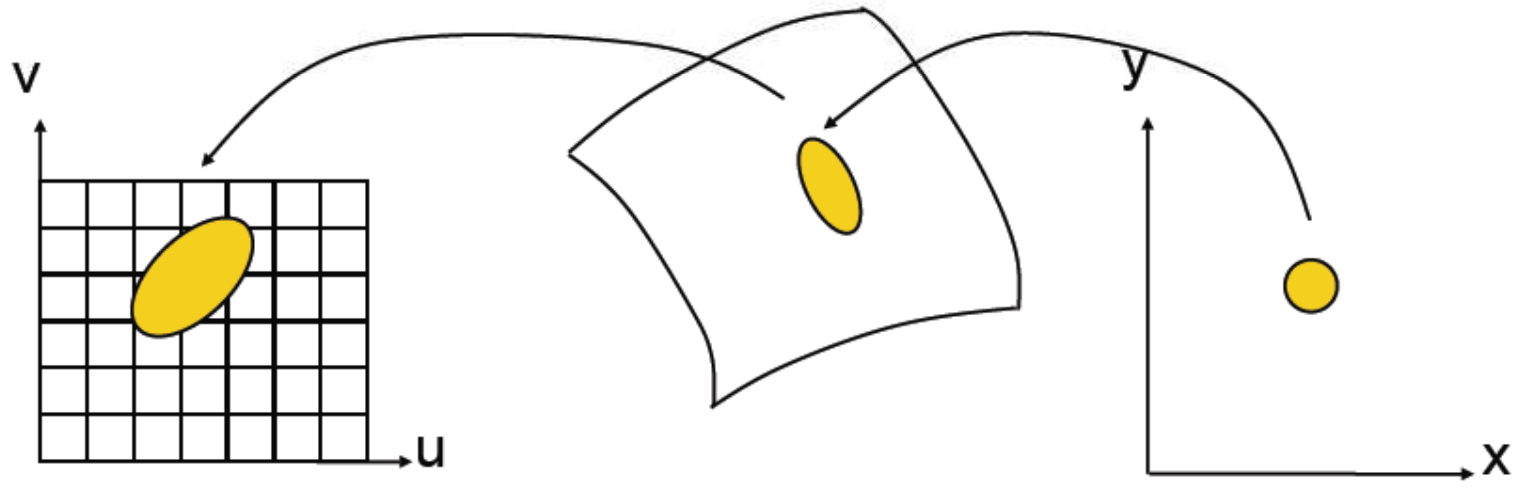
Magnification

- This happens when you zoom really close into a texture mapped polygon or due to perspective projection
- A pixel projects to something smaller than a texel in texture space

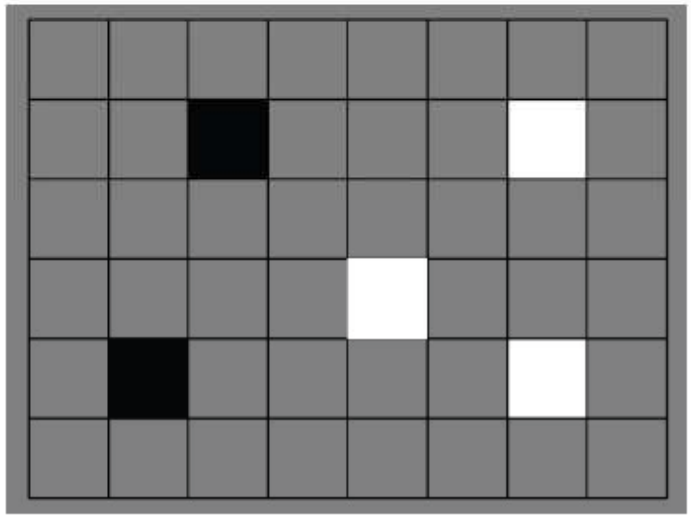


Minification

- This happens when you zoom out or due to perspective foreshortening.
- A pixel projects onto several texels in texture space.



Nearest Neighbor Interpolation



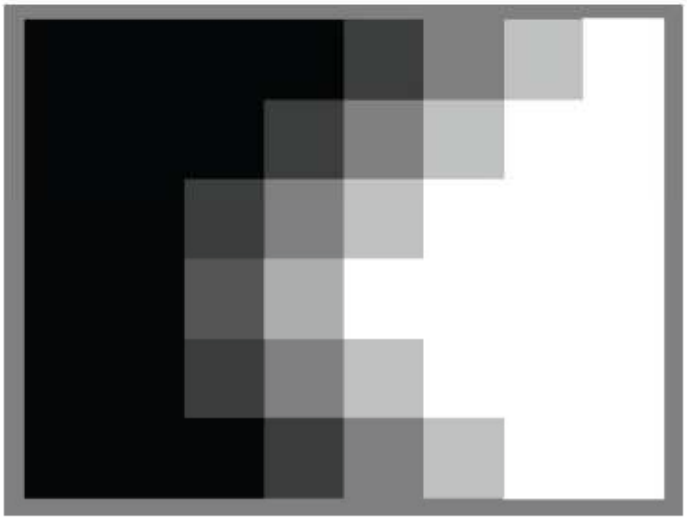
Nearest Neighbor Interpolation

Minification

Magnification

Better Filters

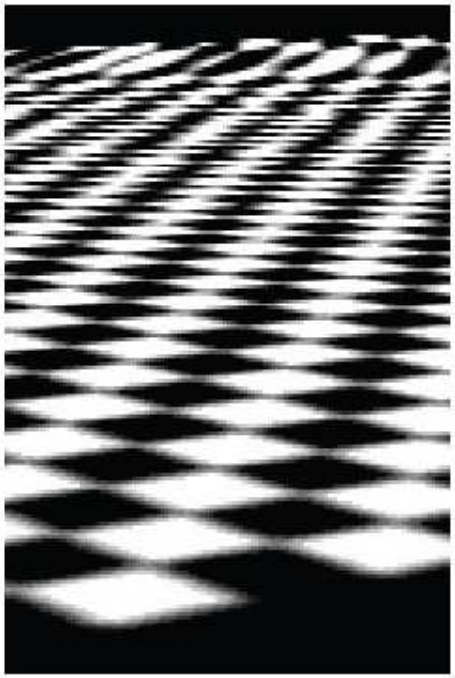
Texture Space



Better Filters

Minification

Magnification

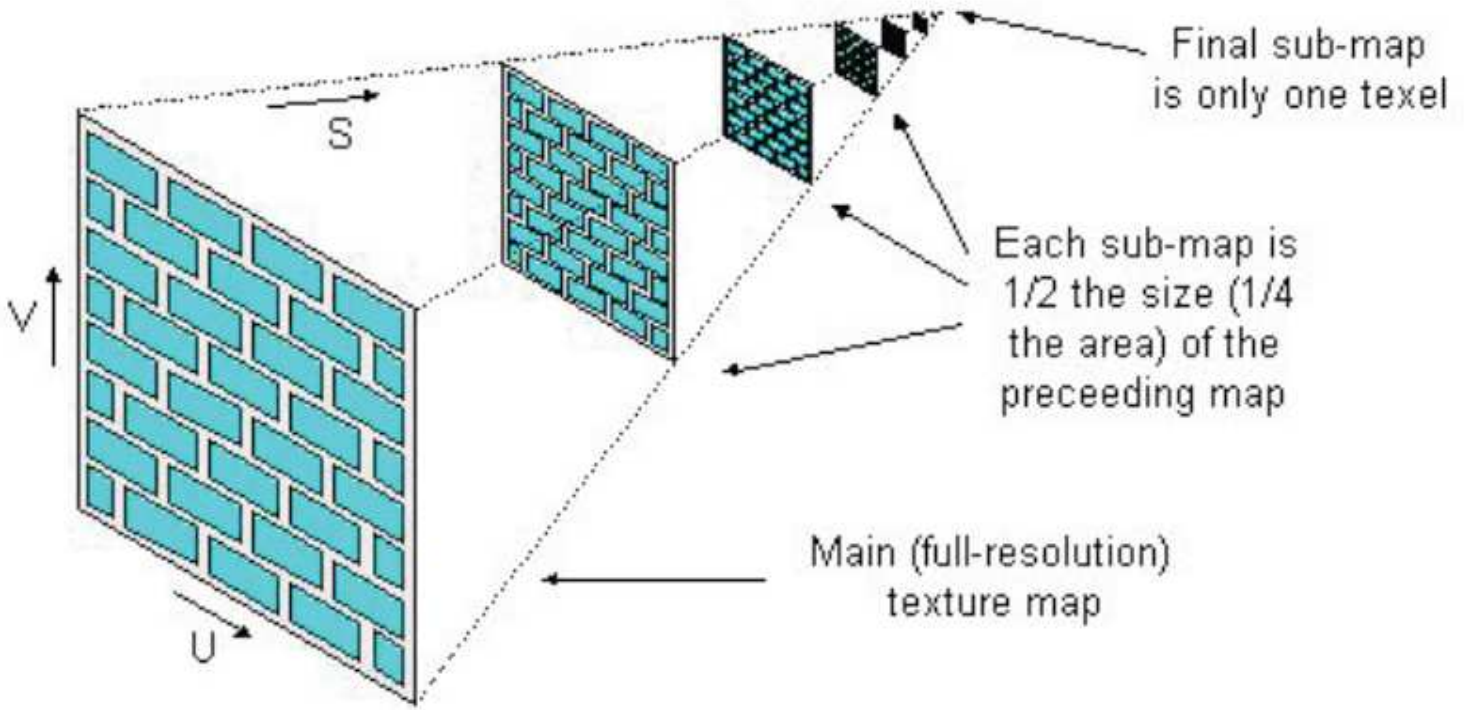


Pre-Filters

- We could perform this filtering during texture mapping
- However, even this is very expensive
- Instead, we can use pre-filtering of the texture prior to rendering

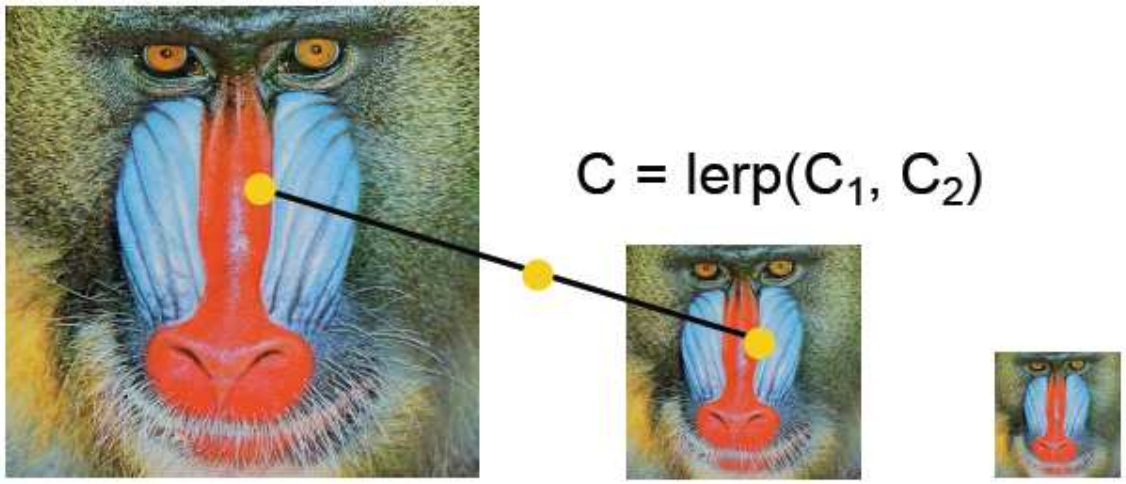
MipMapping

- The basic idea is to construct a pyramid of images that are pre-filtered and down-sampled



MipMap Interpolation

- Sometimes, a pixel really should be filtered with a filter size that is not part of the mipmap
- We can use linear interpolation between mipmap levels to compute the texel color
- If we use a bi-linear filter inside the mip-mapped texture, this is called tri-linear interpolation



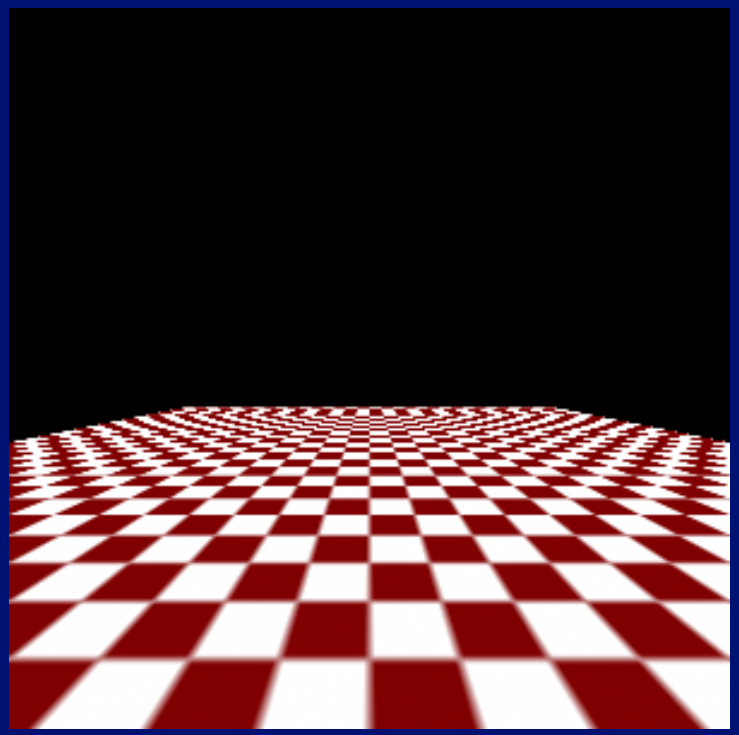
Generate Mipmaps

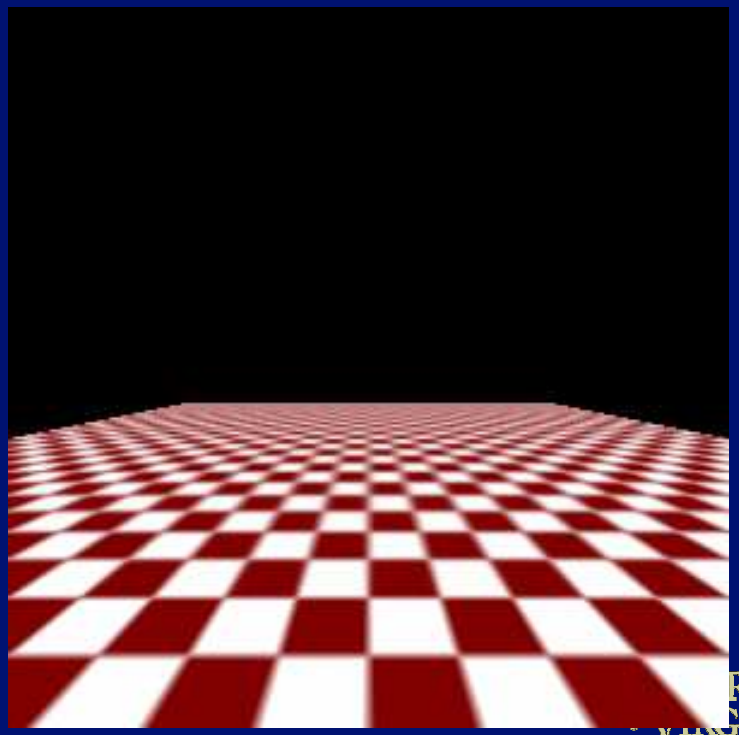
- You can use **`glTexImage2D()`** directly, but it's much simpler to use **`gluBuild2DMipmaps()`**
  - ```
    gluBuild2DMipmaps(GL_TEXTURE_2D, GL_RGB,
    ```
  - ```
        width, height, GL_RGB,
    GL_UNSIGNED_BYTE, mytextureimage);
    ```

- This is a great function! The 2D texture does not even need to be a power of 2!

MIPMAPS

With versus without MIPMAP





Filtering

OpenGL tries to pick best mipmap level

Question: Which texel corresponds to a particular pixel?

GL_NEAREST (Point Sampling)

- Pick the texel with center nearest pixel

GL_LINEAR (Bilinear Sampling)

- Weighted average of 2x2 closest texels

GL_NEAREST_MIPMAP_LINEAR

- Average nearest texels from two mipmap levels

GL_LINEAR_MIPMAP_LINEAR (Trilinear)

- Average two averaged texels from two mipmaps

DEVIL

DevIL

- เป็นไลบรารีสำหรับอ่านและเขียนไฟล์รูปภาพต่างๆ
  - นามสกุล .jpg, .png, .tiff, ฯลฯ
- ติดต่อกับ OpenGL ได้ดี
- ดาวน์โหลด
  - http://openil.sourceforge.net
  - DevIL 1.7.8 SDK for 32-bit Windows
- Unzip แล้ว
  - นำไฟล์ .dll ไปใส่ไว้ใน bin ของที่ไว้ไฟล์สำหรับ CMake
  - นำไฟล์ .lib ไปใส่ไว้ใน lib ของที่ไว้ไฟล์สำหรับ CMake
  - นำไฟล์ .h ต่างๆ ไปใส่ไว้ใน include/IL ของที่ไว้ไฟล์สำหรับ CMake

ไฟล์ที่ต้อง include

- สั่ง
  - #include <IL/il.h>
  - #include <IL/ilu.h>
  - #include <IL/ilut.h>

  ไว้ที่หัวโปรแกรม

ฟังก์ชัน init ต่างๆ

- มีไว้เพื่อให้ DevIL ทั้งค่าเริ่มต้นของตนเอง
- ในฟังก์ชัน main สั่ง
  - ilInit();
  - iluInit();

  ก่อนเรียกฟังก์ชันอื่นๆ ทั้งหมด
- แล้วเรียก
  - ilutInit();

  หลังเรียก glutInit(…);

Image Object

- DevIL มีระบบ “รูป” คล้ายกับระบบ texture ของ OpenGL
  - สร้างด้วย ilGenImages
  - กำหนดด้วย ilBindImage
  - อ่านรูปจากไฟล์ด้วย ilLoadImage
  - ลบด้วย ilDeleteImages

ilGenImages

- ILvoid ilGenImages(
  ILsizei Num, ILuint *ids);
  - สร้าง image object หลายๆ อัน
  - argument แรกคือจำนวน image object ที่ต้องการสร้าง
  - argument ที่สองคืออะไรสำหรับใส่ “ชื่อ” ของ image object
  - การสร้าง image object เพียงแค่อันเดียว
    - ILuint id;
    - ilGenImages(1, &id);
  - การสร้าง image object หลายอัน
    - ILuint id[3];
    - ilGenImages(3, id);

ilBindImage

- ILvoid ilBindImage(ILuint id)
  - ใช้เซต image object ปัจจุบันให้เป็น image object ที่มีชื่อที่กำหนด
  - ตัวอย่าง
    - ILuint id;
    - ilGenImages(1, &id);
    - ilBindImage(id);

ilLoadImage

- ILboolean ilLoadImage(const char *FileName)
  - ใช้สำหรับอ่านรูปจากไฟล์ที่กำหนดชื่อให้มายัง image object ปัจจุบันที่ bind ไว้แล้ว
  - argument แรกคือชื่อไฟล์
    - DevIL จะรู้ว่าเป็นไฟล์ชนิดใดโดยอัตโนมัติ
  - คืนค่า “จริง” ถ้าสามารถอ่านรูปได้และคืนค่า “เท็จ” ถ้าอ่านไม่สำเร็จ
  - ตัวอย่าง
    - ILuint id;
    - ilGenImages(1, &id);
    - ilBindImage(id);
    - ilLoadImage(“test.jpg”);

ilDeleteImages

- ILvoid ilDeleteImages(
  ILsizei Num, ILuint *ids);
  - ลบ image object หลายๆ อัน
  - argument แรกคือจำนวน image object ที่ต้องการลบ
  - argument ที่สองคืออะไรสำหรับใส่ “ชื่อ” ของ image object
  - การลบ image object เพียงแค่อันเดียว
    - ILuint id;
    - …
    - ilDeleteImages(1, &id);
  - การสร้าง image object หลายอัน
    - ILuint id[3];
    - ilGenImages(3, id);
    - …
    - ilDeleteImages(3, id);

ilConvertImage

- ILboolean ilConvertImage(
  ILenum DestFormat, ILenum DestType)
  - ใช้สำหรับแปลงข้อมูลรูปภาพที่อยู่ในภาพที่ bind ไว้เป็นรูปแบบอื่น
  - DestFormat คือรูปแบบสีของแต่ละ pixel
    - IL_RGB, IL_RGBA, IL_BGR, IL_BGRA, IL_LUMINANCE, IL_COLOUR_INDEX
  - DestType คือชนิดข้อมูลที่ใช้เก็บสีแต่ละช่อง
    - IL_BYTE, IL_UNSIGNED_BYTE, IL_SHORT , IL_UNSIGNED_SHORT, IL_INT, IL_UNSIGNED_INT, IL_FLOAT, IL_DOUBLE

การส่งข้อมูลรูปภาพเข้า OpenGL Texture

- เรียก glTexImage2D โดยให้ข้อมูลต่างๆ จาก Image Object
  - internalFormat ให้ป้อน ilGetInteger(IL_IMAGE_BPP)
  - width ให้ป้อน ilGetInteger(IL_IMAGE_WIDTH)
  - height ให้ป้อน ilGetInteger(IL_IMAGE_HEIGHT)
  - format ให้ป้อน ilGetInteger(IL_IMAGE_FORMAT)
  - type ให้ป้อน ilGetInteger(IL_IMAGE_TYPE)
  - data ให้ป้อน ilGetData()

ilGetData

- ILubyte* ilGetData(ILvoid)
  - คืนข้อมูลของ image object ที่ bind ไว้เป็นอะเรย์ของ unsigned byte

ตัวอย่างการใช้งาน

```
glGenTextures(1, &tex0);
glBindTexture(GL_TEXTURE_2D, tex0);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR_MIPMAP_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_LINEAR);

ILuint image0;
ilGenImages(1, &image0);
ilBindImage(image0);
char imageName[] = "../../data/yellow-flowers.jpg";
ilLoadImage(imageName);
ilConvertImage(IL_RGB, IL_UNSIGNED_BYTE);

gluBuild2DMipmaps(GL_TEXTURE_2D, ilGetInteger(IL_IMAGE_BPP),
    ilGetInteger(IL_IMAGE_WIDTH), ilGetInteger(IL_IMAGE_HEIGHT),
ilGetInteger(IL_IMAGE_FORMAT), ilGetInteger(IL_IMAGE_TYPE), ilGetData());

ilDeleteImages(1, &image0);
```