CSCI 420 Computer Graphics
Lecture 12

Texture Mapping

Texture Mapping + Shading
Filtering and Mipmaps
Non-color Texture Maps
[Angel Ch. 7]

Oded Stein
University of Southern California

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
- Efficiently supported in hardware

Trompe L'Oeil ("Deceive the Eye")

Jesuit Church, Vienna, Austria

- Windows and columns in the dome are painted, not a real 3D object
- Similar idea with texture mapping:

  Rather than modeling the intricate 3D geometry, replace it with an image !

You can get away with very few polygons

Black & White, 2001

Map textures to surfaces

texture map

an image

image mapped
to a 3D polygon

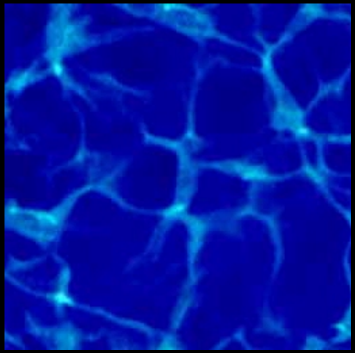
The polygon can have
arbitrary size, shape and
3D position

The texture

- Texture is a bitmap image
  - Can use an image library to load image into memory
  - Or can create images yourself within the program

- 2D array:
  `unsigned char texture[height][width][4]`

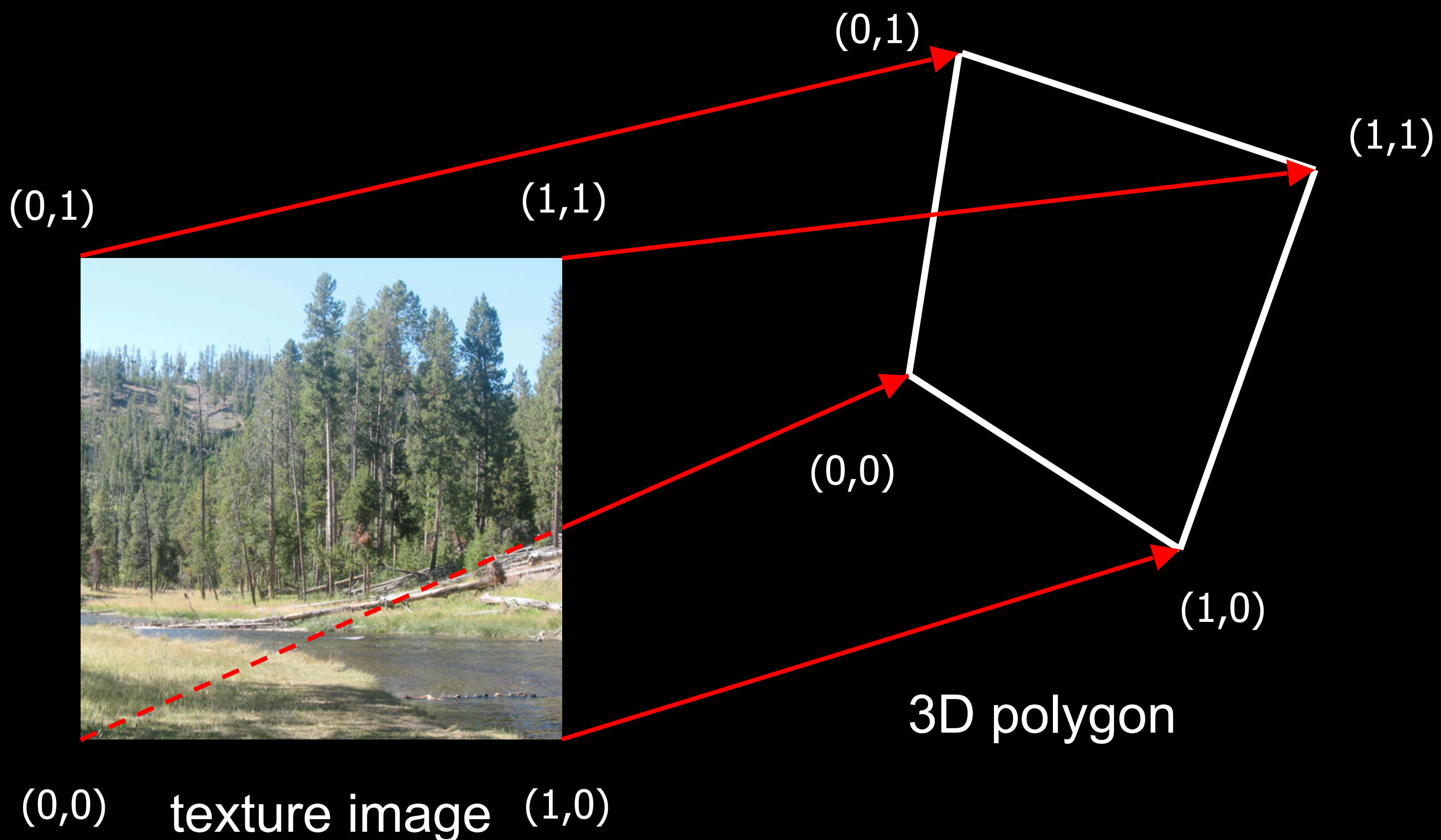
- Or unrolled into 1D array:
  `unsigned char texture[4*height*width]`



- Pixels of the texture are called *texels*

- Texel coordinates (s,t) scaled to [0,1] range

Texture map



Texture map

(0,1) (1,1)

(0,0) texture image (1,0)

(0,1)

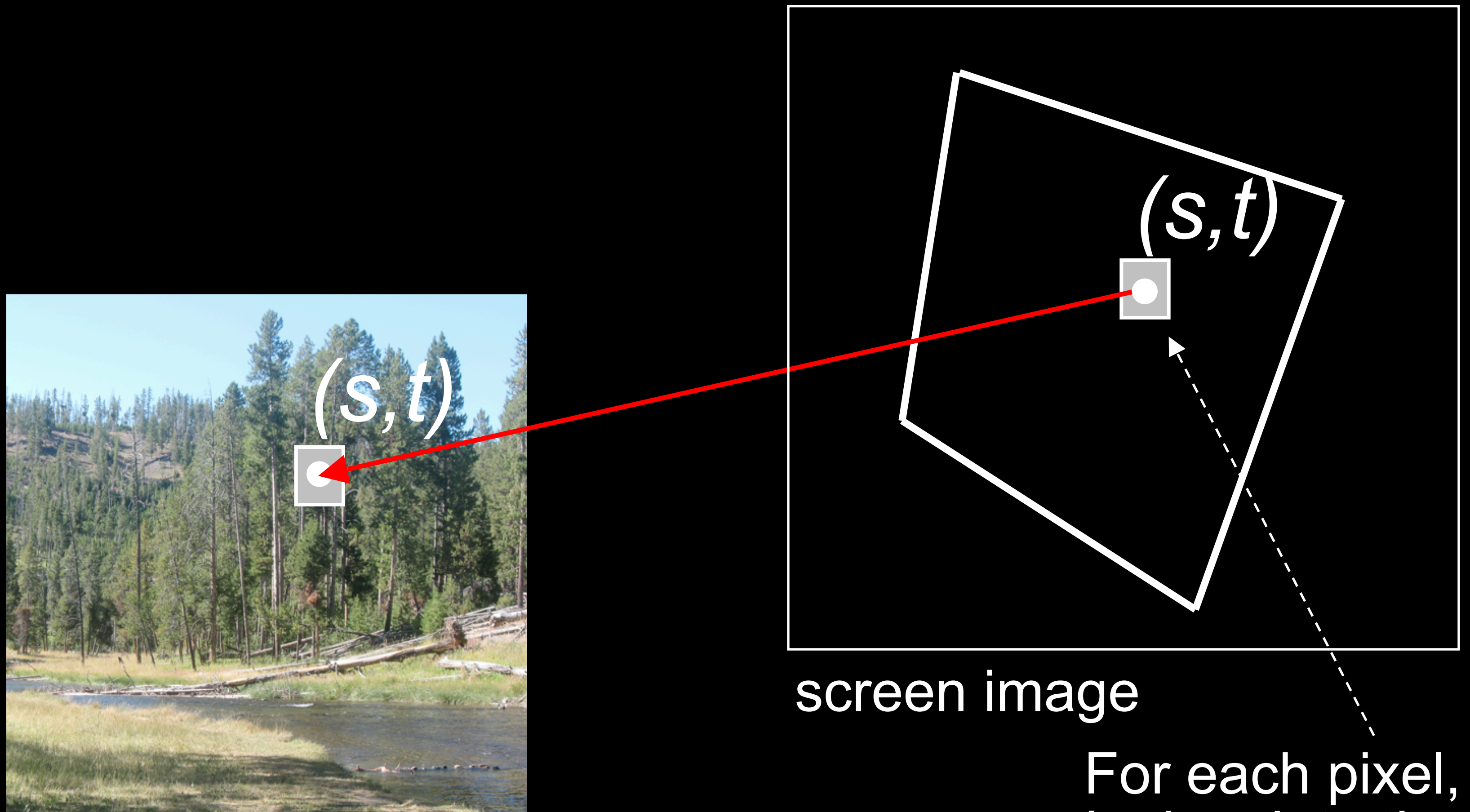
(1,1)

(0,0)

(1,0)

3D polygon

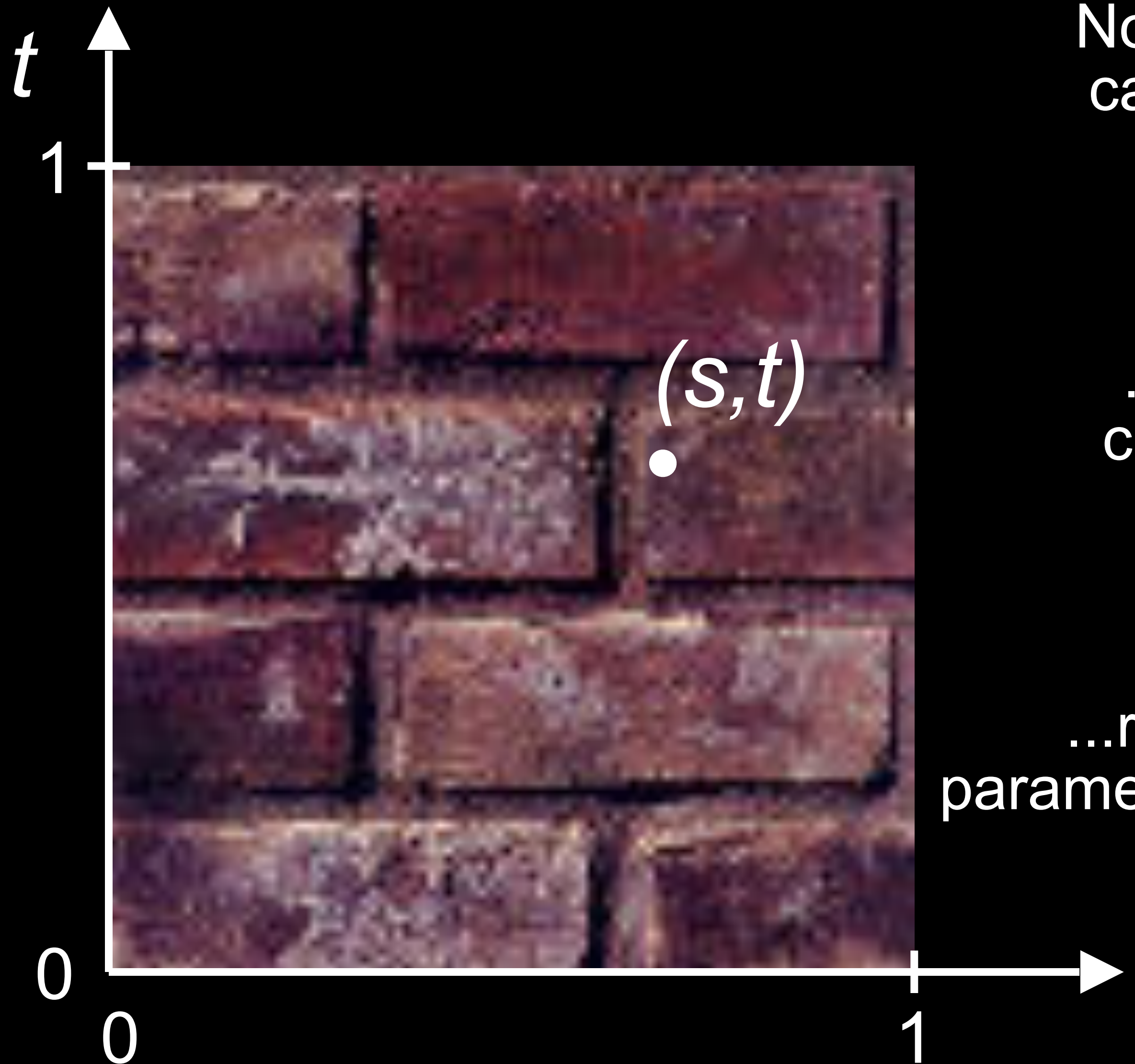
Texture coordinates



texture image

For each pixel, lookup into the texture image to obtain color.

The "st" coordinate system

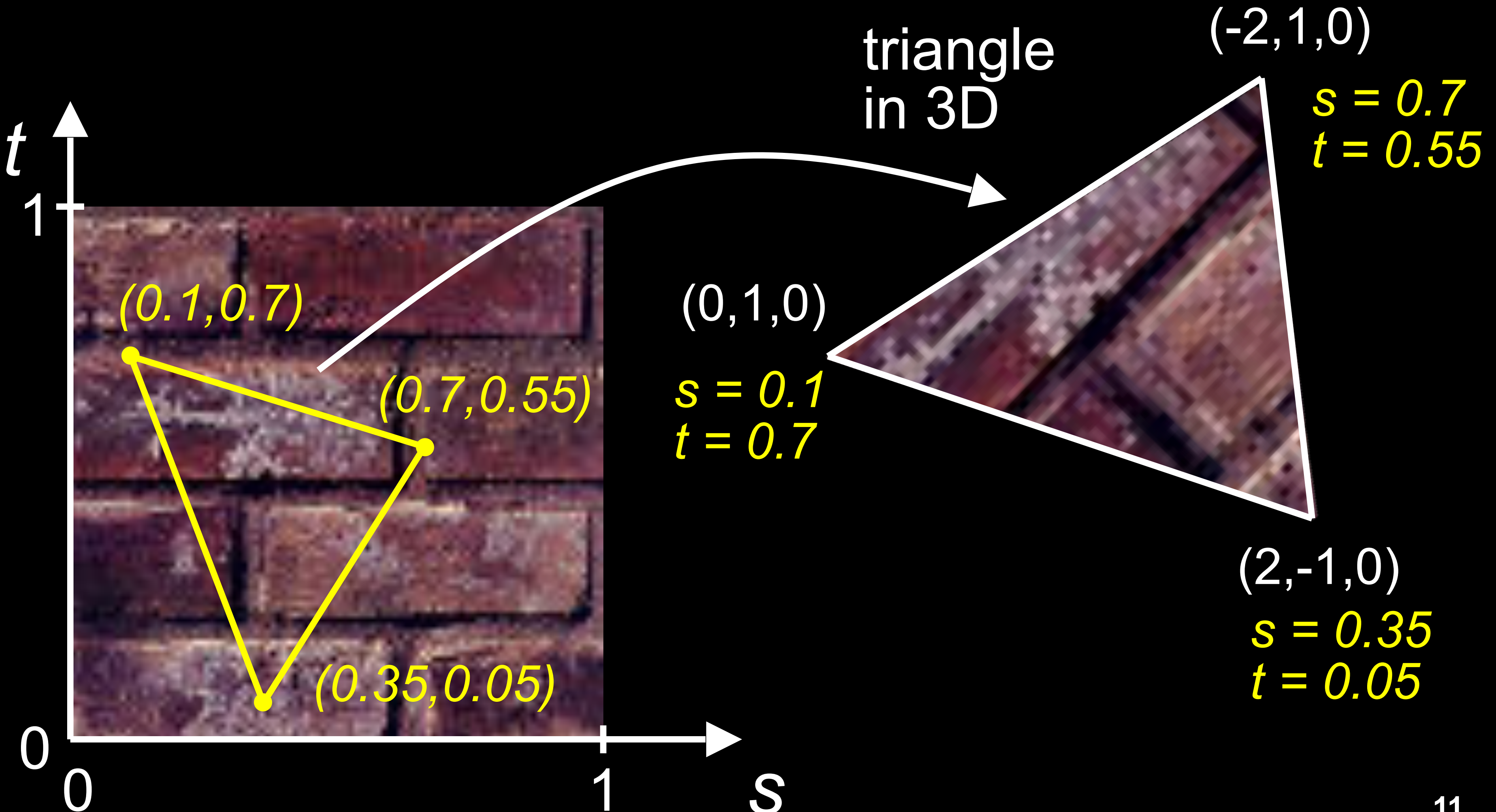


Note: also called UV space

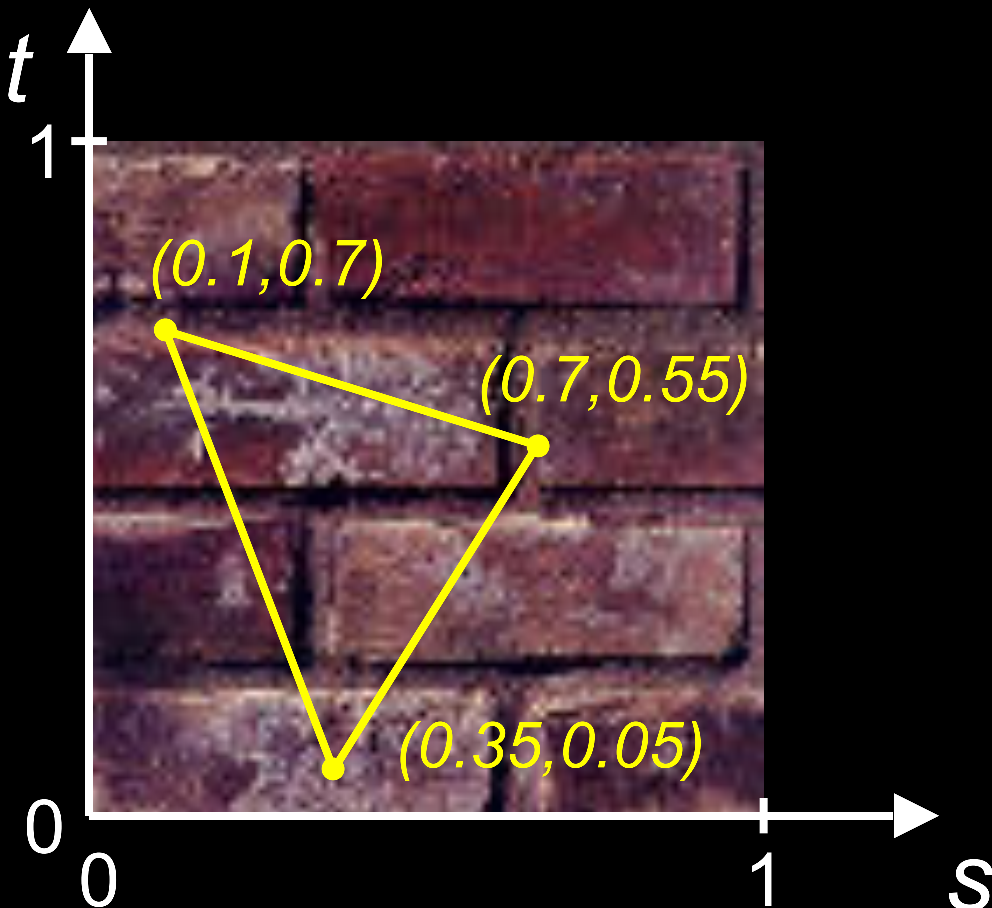
...usually called UV space

...remember parametrization?

Texture mapping: key slide



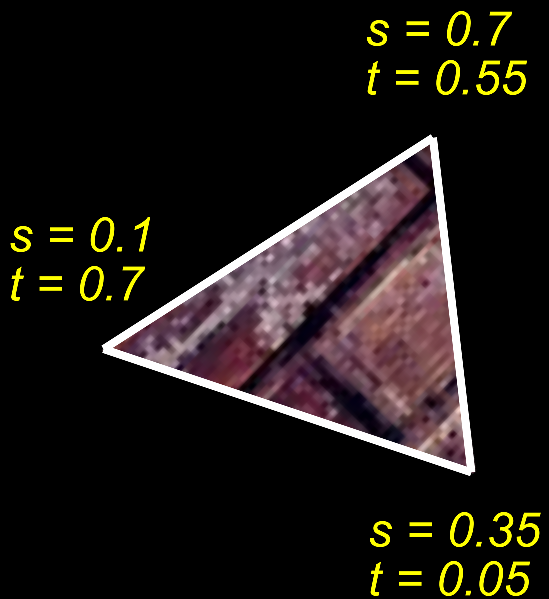
Texture mapping: key slide



- This really is the only thing you need to define a texture map (UV map)
- We need to know where to place every vertex of the triangle in the UV plane.

Specifying texture coordinates in OpenGL (core profile)

- Use VBO
- Either create a separate VBO for texture coordinates, or put them with vertex positions into one VBO
  - (do yourself a favor and create a separate VBO)
- If separate VBO: separate input variable in vertex shader
  - **Do you need to apply modelview/projection to the texture coordinates?**

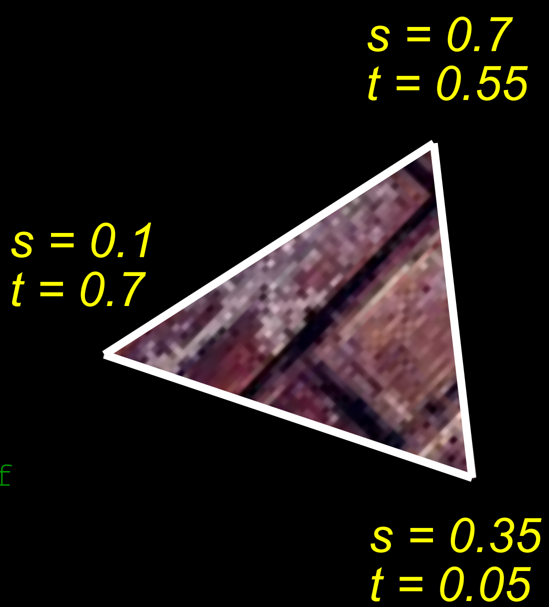


...no

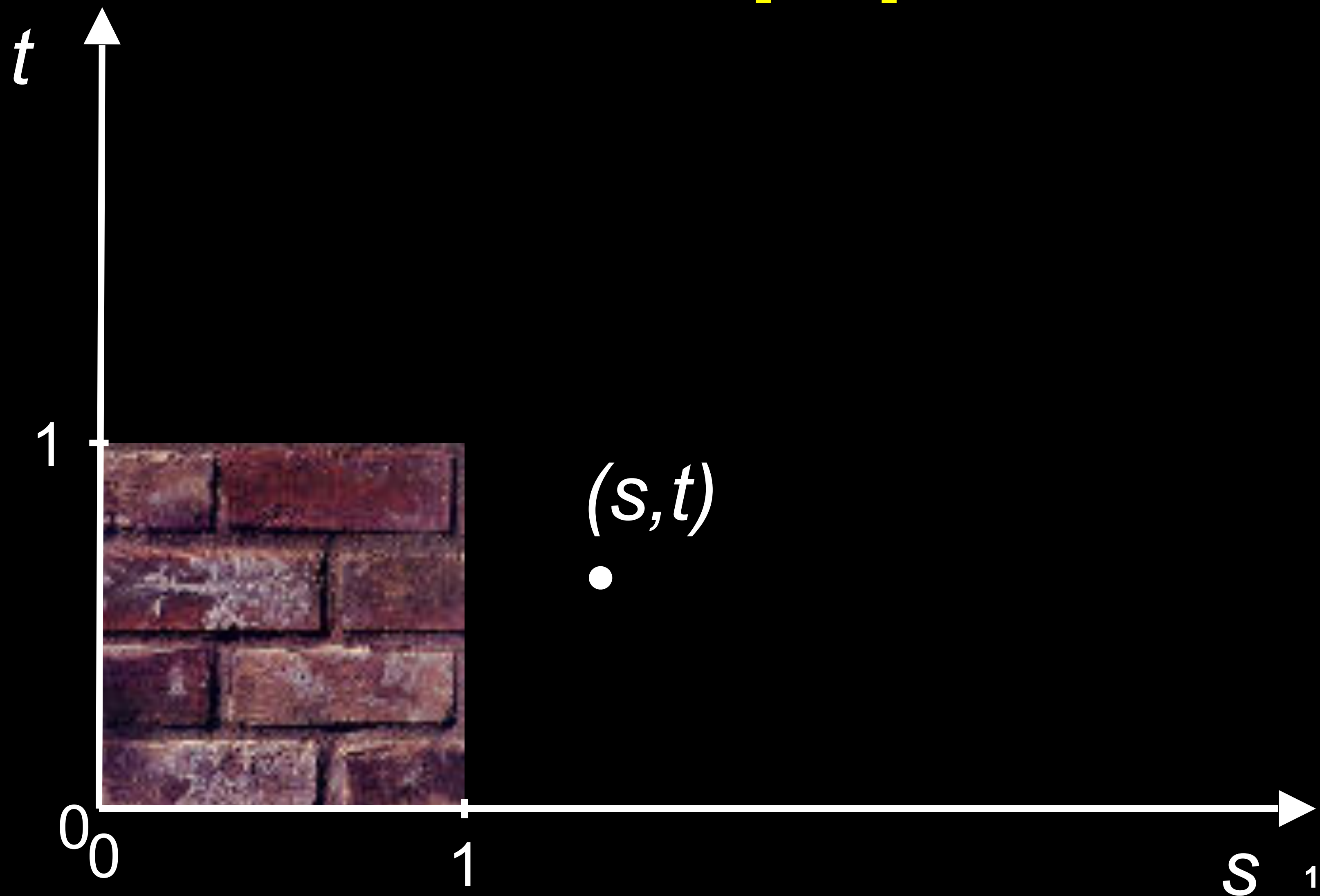
Specifying texture coordinates in OpenGL (compatibility profile)

- Use `glTexCoord2f(s,t)`
- State machine: Texture coordinates remain valid until you change them
- Example (from the previous slide) :

```
glEnable(GL_TEXTURE_2D); // turn texture mapping on
glBegin(GL_TRIANGLES);
  glTexCoord2f(0.35,0.05); glVertex3f(2.0,-1.0,0.0);
  glTexCoord2f(0.7,0.55); glVertex3f(-2.0,1.0,0.0);
  glTexCoord2f(0.1,0.7); glVertex3f(0.0,1.0,0.0);
glEnd();
glDisable(GL_TEXTURE_2D); // turn texture mapping off
```

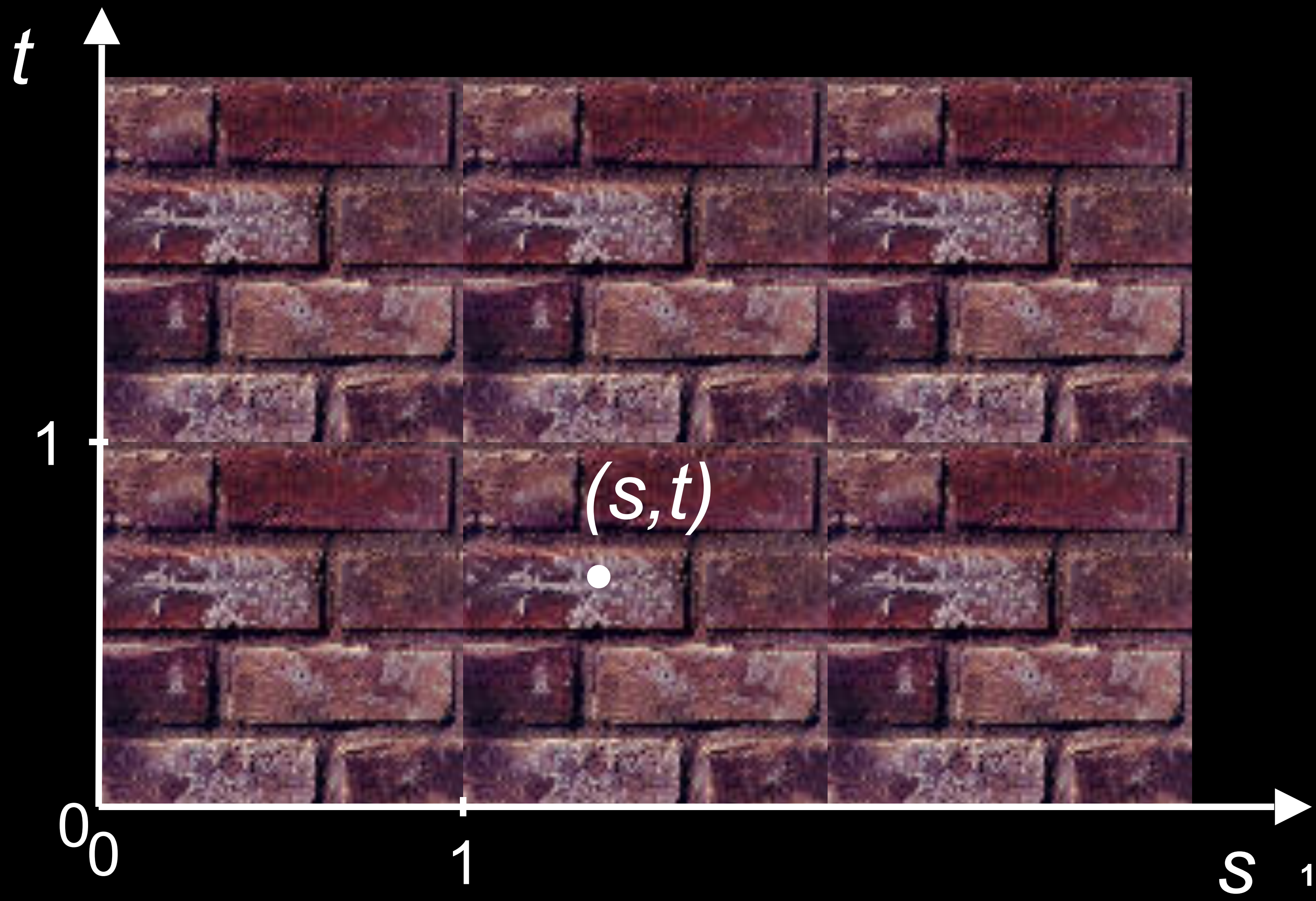


What if texture coordinates are outside of [0,1] ?



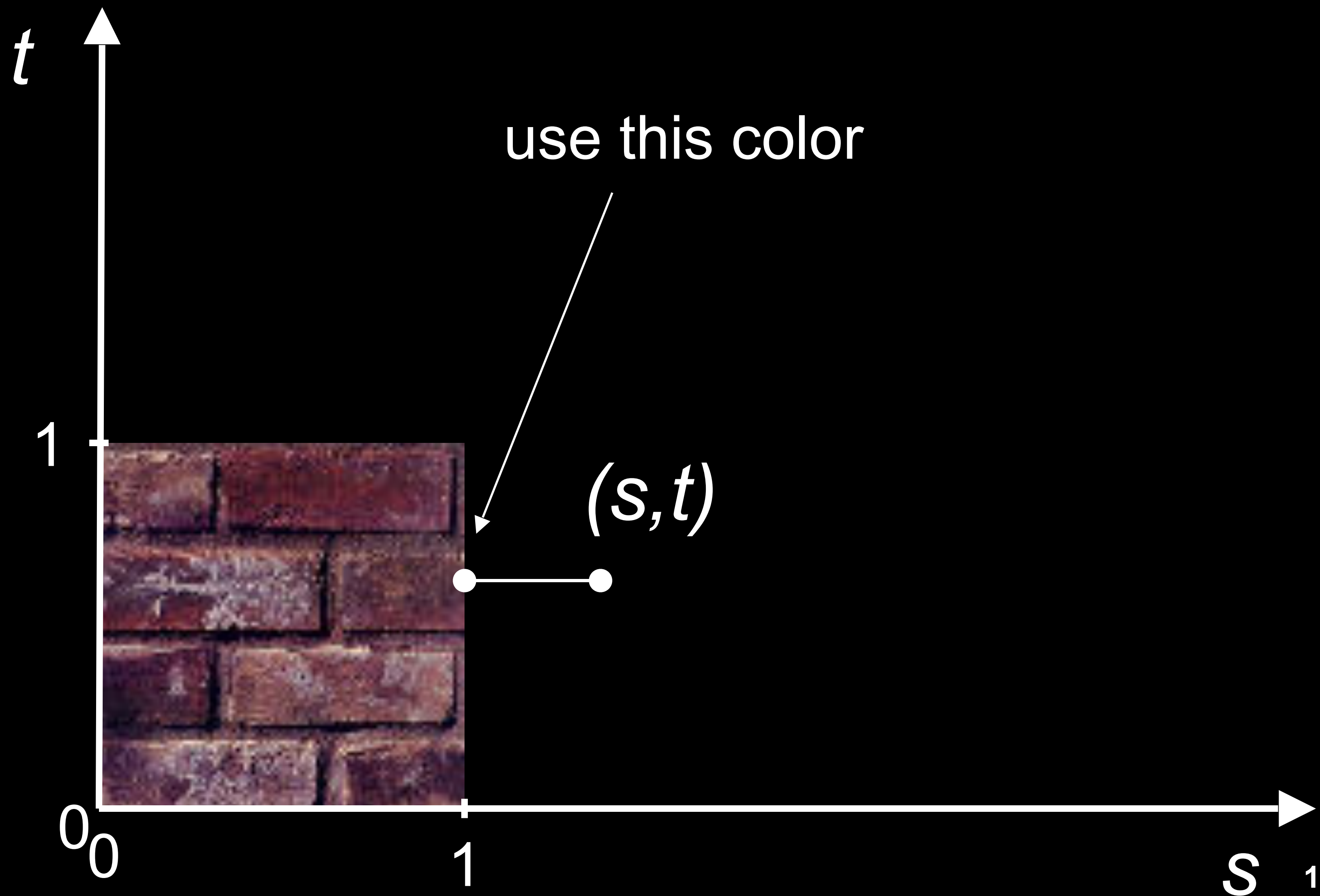
Solution 1: Repeat texture

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT)
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT)
```

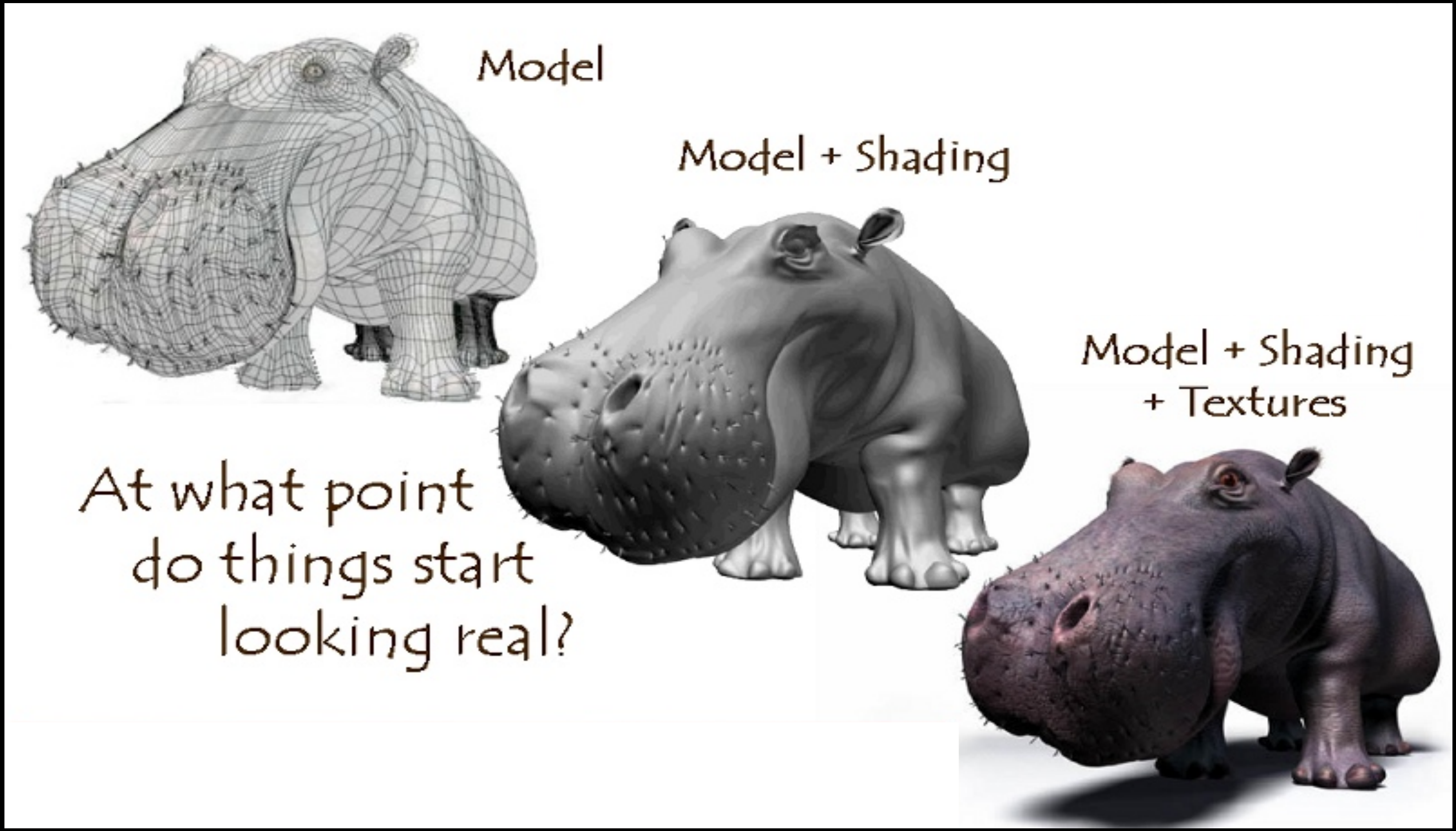


Solution 2: Clamp to [0,1]

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE)
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE)
```



Combining texture mapping and shading



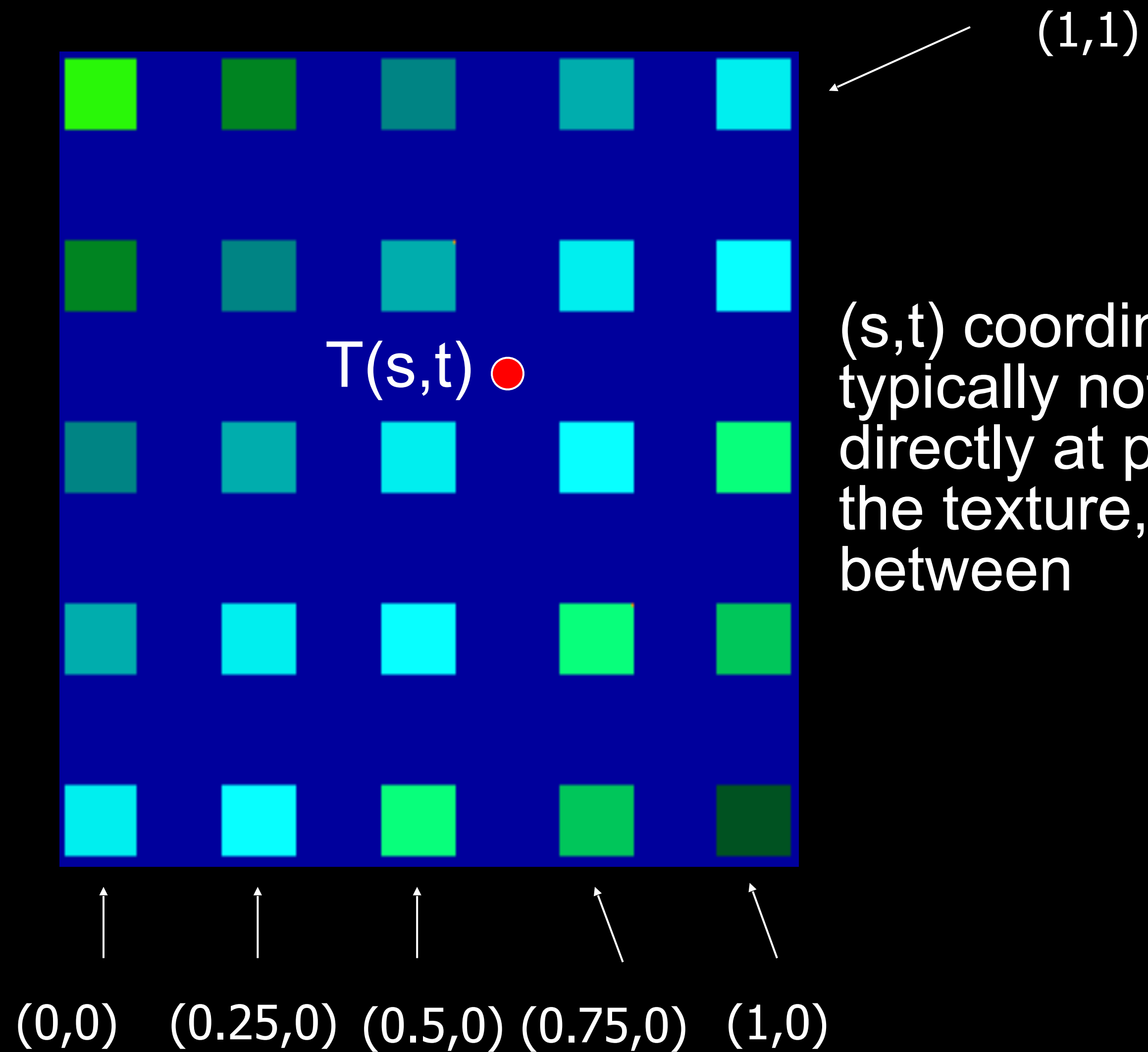
Source: Jeremy Birn

Outline

- Introduction
- Filtering and Mipmaps
- Non-color texture maps
- Texture mapping in OpenGL

Texture interpolation

5 x 5 texture



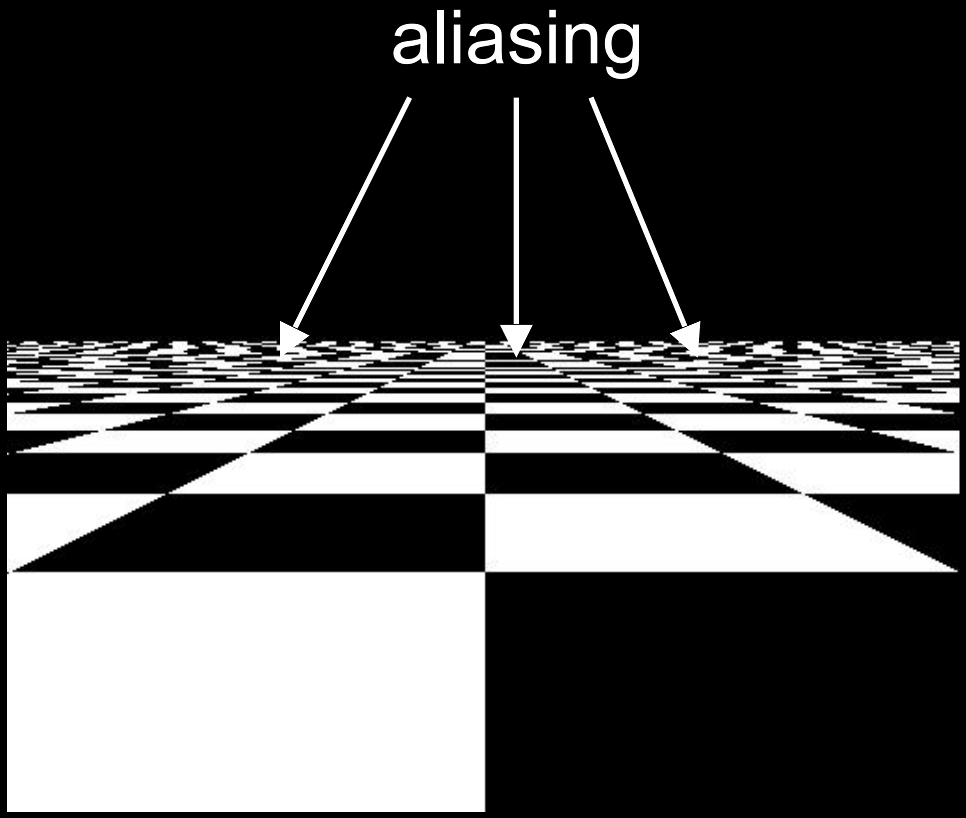
(s,t) coordinates typically not directly at pixel in the texture, but in between

Texture interpolation

- (s,t) coordinates typically not directly at pixel in the texture, but in between
- Solutions:
  - 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)`

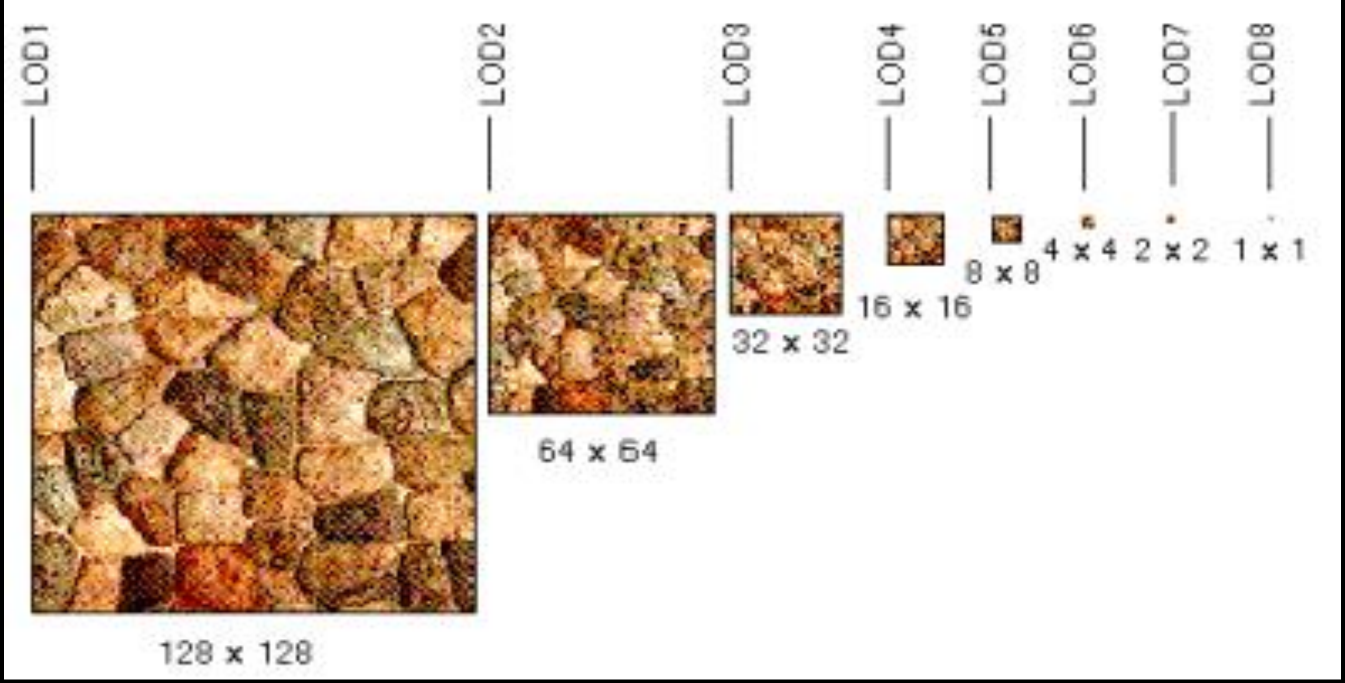
Filtering

- Texture image is shrunk in distant parts of the image
- This leads to aliasing
- Can be fixed with *filtering*
  - bilinear in space
  - trilinear in space and level of detail (mipmapping)



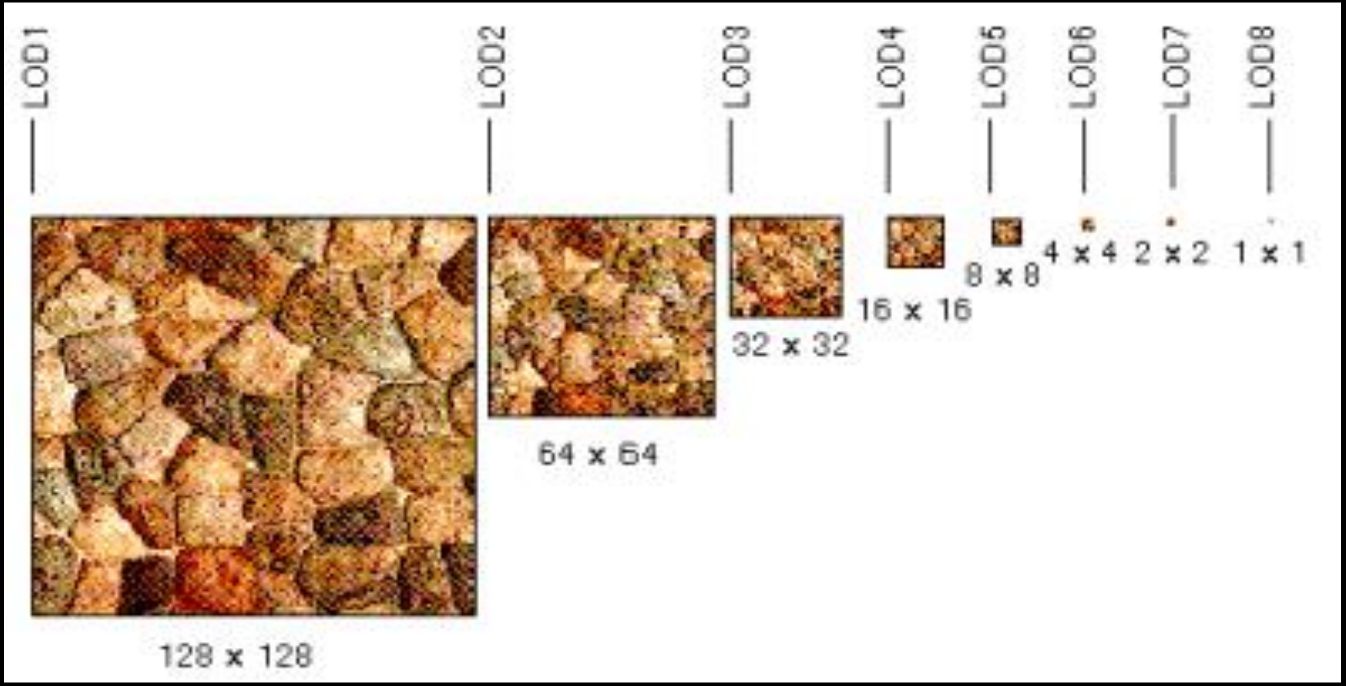
Mipmapping

- Pre-calculate how the texture should look at various distances, then use the appropriate texture at each distance
- Reduces / fixes the aliasing problem



Mipmapping

- Each mipmap (each image below) represents a level of depth (LOD).
- Decrease image 2x at each level



Mipmapping in OpenGL

- Generate mipmaps automatically
(for the currently bound texture):

  Core profile:
```
glGenerateMipmap(GL_TEXTURE_2D);
```

  Compatibility profile:
```
gluBuild2DMipmaps(GL_TEXTURE_2D,
  components, width, height, format, type, data)
```

- Must also instruct OpenGL to use mipmaps:

```
glTexParameteri(GL_TEXTURE_2D,
    GL_TEXTURE_MIN_FILTER,
    GL_LINEAR_MIPMAP_LINEAR)
```

Outline

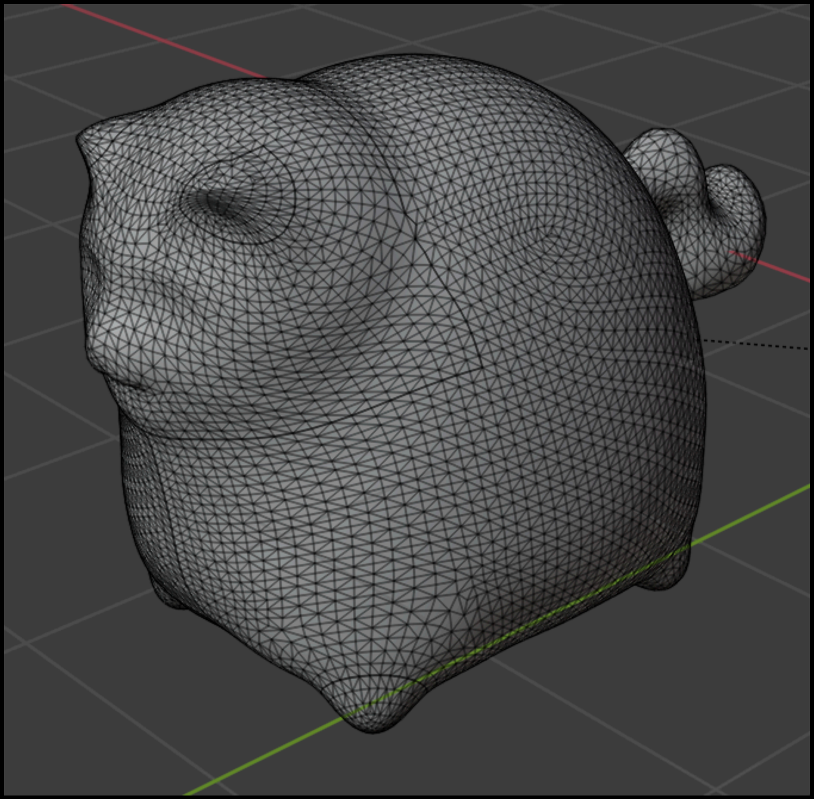
- Introduction
- Filtering and Mipmaps
- Non-color texture maps
- Texture mapping in OpenGL

Textures do not have to represent color

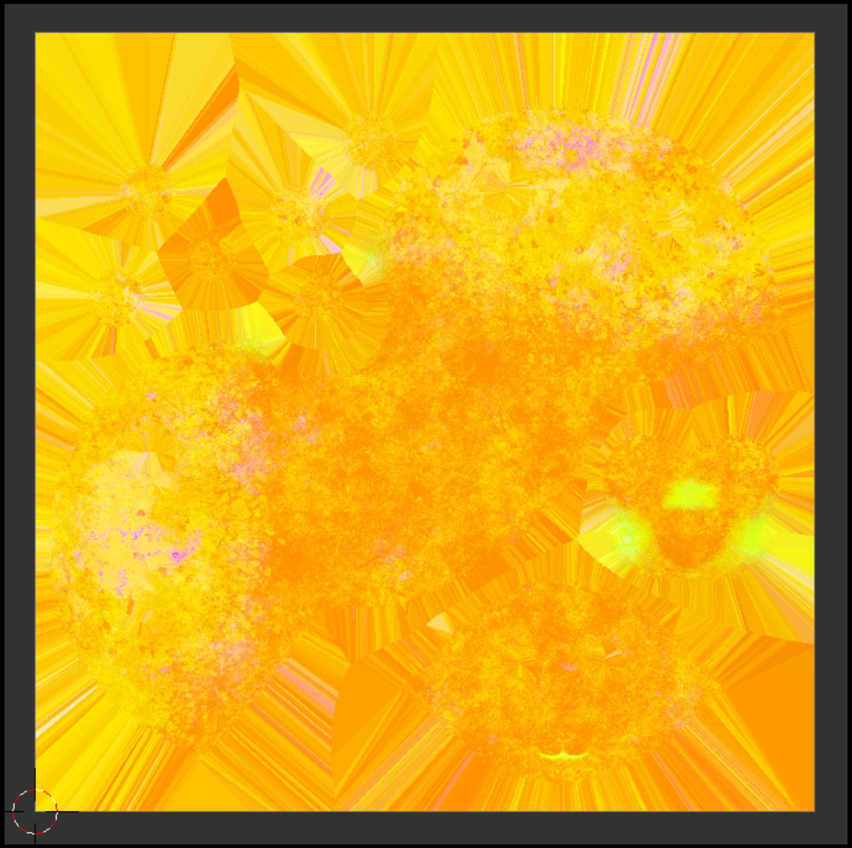
- Specularity (patches of shininess)
- Transparency (patches of clearness)
- Normal vector changes (bump maps)
- Reflected light (environment maps)
- Shadows
- Changes in surface height (displacement maps)
- (secret bonus: non-graphical data)

Specular maps

- (This is really a roughness map. Roughness is related to, but not exactly the same as specularity)



mesh

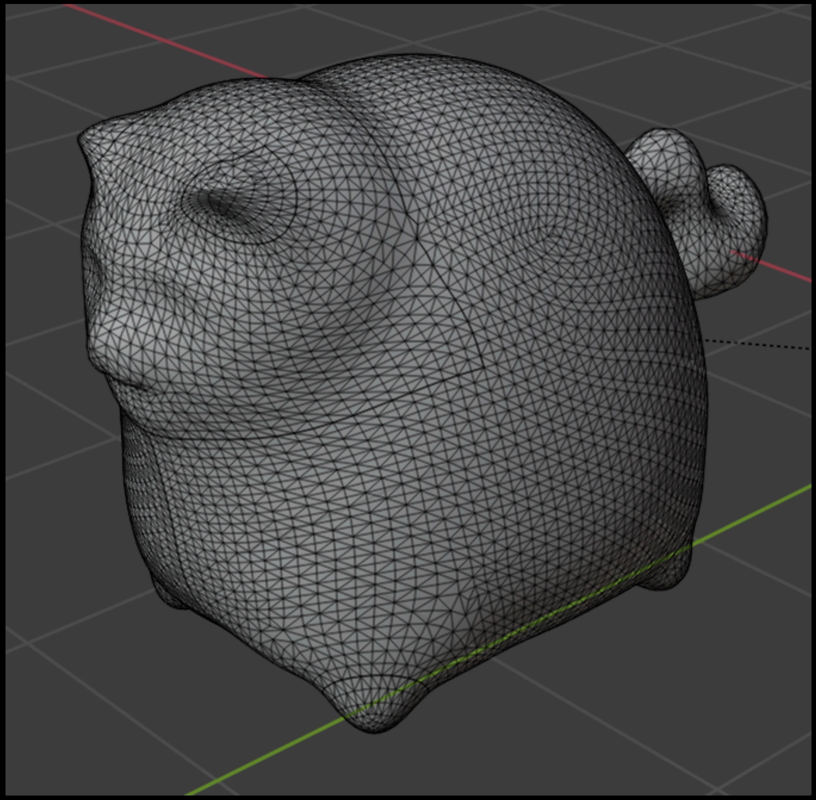


roughness texture

Model by warkama on sketchfab

Specular maps

- (This is really a roughness map. Roughness is related to, but not exactly the same as specularity)



mesh

roughness texture + UV map

Model by warkama on sketchfab

Example: Far Cry 4 (low mapping setting)

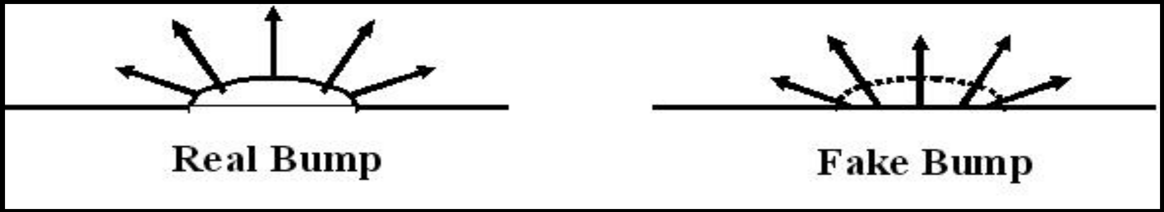
Note the low detail on the weapon.

Example: Far Cry 4 (high mapping setting)

Note the high detail on the weapon, due to specular mapping.

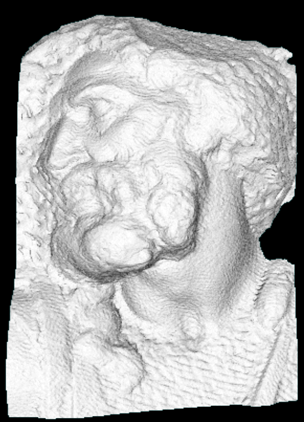
Bump mapping

- How do you make a surface look *rough*?
  - Option 1: model the surface with many small polygons
  - Option 2: perturb the normal vectors before the shading calculation
    - » Fakes small displacements above or below the true surface
    - » The surface doesn't actually change,
      but shading makes it look like there are irregularities!
    - » A texture stores information about the "fake" height of the surface

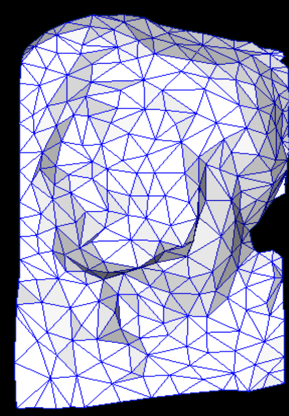


Bump mapping

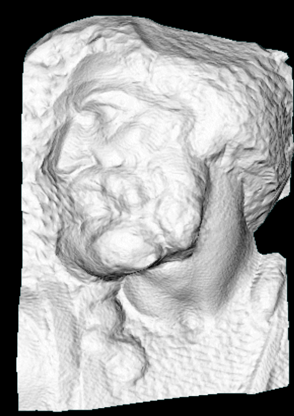
- We can perturb the normal vector without having to make any actual change to the shape.
- This illusion can be seen through—how?



Original model
(5M)

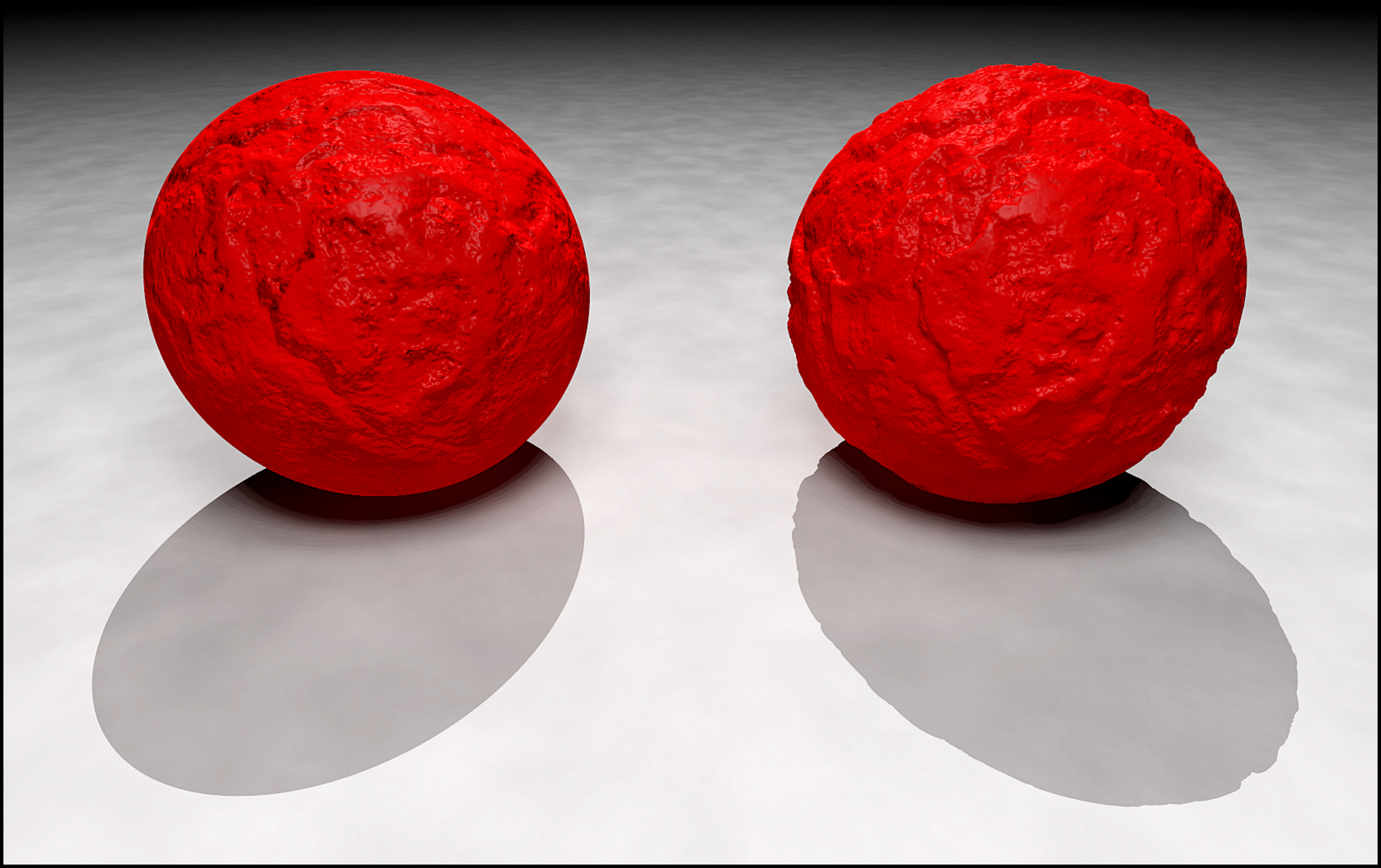


Simplified
(500)



Simple model with
bump map

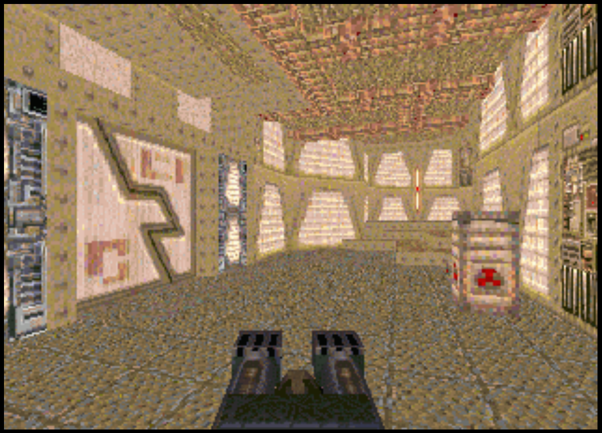
Bump vs Displacement Mapping



Left: bump mapping   Right: displacement mapping

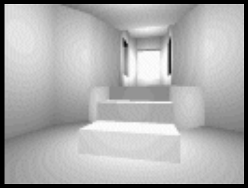
Light Mapping

- *Quake* uses *light maps* in addition to texture maps. Texture maps are used to add detail to surfaces, and light maps are used to store pre-computed illumination. The two are multiplied together at run-time, and cached for efficiency.



Texture Map Only

Texture + Light Map



Light Map

Example: Far Cry 4 (low mapping setting)

Note the low detail on the walls, due to low-resolution displacement mapping.

Example: Far Cry 4 (high mapping setting)

Note the high detail on the walls, due to high-resolution displacement mapping.

Outline

- Introduction
- Filtering and Mipmaps
- Non-color texture maps
- Texture mapping in OpenGL

OpenGL Texture Mapping (Core Profile)

- During initialization:
  1. Read texture image from file into an array in memory, or generate the image using your program
  2. Initialize the texture (`glTexImage2D`)
  3. Specify texture mapping parameters:
     - » Repeat/clamp, filtering, mipmapping, etc.
  4. Make VBO for the texture coordinates
  5. Create VAO
- In display():
  1. Bind VAO
  2. Select the texture unit, and texture (using `glBindTexture`)
  3. Render (e.g., `glDrawArrays`)

Read texture image from file into an array in memory

- Can use our ImageIO library

- ```
  ImageIO * imageIO = new ImageIO();
  if (imageIO->loadJPEG(imageFilename) != ImageIO::OK)
  {
     cout << "Error reading image " << imageFilename << "." << endl;
     exit(EXIT_FAILURE);
  }
  ```

- See starter code for hw2
  ```

Initializing the texture

- Do once during initialization, for each texture image in the scene, by calling `glTexImage2D`

- The dimensions of texture images must be a multiple of 4 (Note: they do NOT have to be a power of 2)

- Can load textures dynamically if GPU memory is scarce:

  Delete a texture (if no longer needed) using `glDeleteTextures`

glTexImage2D

- `glTexImage2D(GL_TEXTURE_2D, level, internalFormat, width, height, border, format, type, data)`

- GL_TEXTURE_2D: specifies that it is a 2D texture
- Level: used for specifying levels of detail for mipmapping (default: 0)
- InternalFormat
  - Often: GL_RGB or GL_RGBA
  - Determines how the texture is stored internally
- Width, Height
  - The size of the texture must be a multiple of 4
- Border (often set to 0)
- Format, Type
  - Specifies what the input data is (GL_RGB, GL_RGBA, …)
  - Specifies the input data type (GL_UNSIGNED_BYTE, GL_BYTE, …)
  - Regardless of Format and Type, OpenGL converts the data to internalFormat
- Data: pointer to the image buffer

Texture Initialization

Global variable:

```
GLUint texHandle;
```

During initialization:

```
// create an integer handle for the texture
glGenTextures(1, &texHandle);

int code = initTexture("sky.jpg", texHandle);
if (code != 0)
{
  printf("Error loading the texture image.\n");
  exit(EXIT_FAILURE);
}
```

Function initTexture() is given in the starter code for hw2.

Texture Shader: Vertex Program

```
#version 150

in vec3 position;
in vec2 texCoord;
```

input vertex position
and texture coordinates

```
out vec2 tc;
```

output texture coordinates; they will be passed to
the fragment program (interpolated by hardware)

```
uniform mat4 modelViewMatrix;
uniform mat4 projectionMatrix;
```

transformation matrices

```
void main()
{
  // compute the transformed and projected vertex position (into gl_Position)
  gl_Position = projectionMatrix * modelViewMatrix * vec4(position, 1.0f);
  // pass-through the texture coordinate
  tc = texCoord;
}
```

Texture Shader: Fragment Program

```
#version 150

in vec2 tc; // input tex coordinates (computed by the interpolator)
out vec4 c; // output color (the final fragment color)
uniform sampler2D textureImage; // the texture image

void main()
{
  // compute the final fragment color,
  // by looking up into the texture map
  c = texture(textureImage, tc);
}
```

Setting up the texture coordinates

During initialization:

```
// Prepare the texture coordinates (the "UV"s).
float * uvs = (float*) malloc (sizeof(float) * numVertices * 2);
// Write into uvs here:
// …

// Put the texture coordinates into a VBO.
// 2 values per vertex, namely u and v.
VBO * vboUVs = new VBO(numVertices, 2, uvs, GL_STATIC_DRAW);

// Connect the shader variable "texCoord" to the VBO.
vao->ConnectPipelineProgramAndVBOAndShaderVariable(
    pipelineProgram, vboUVs, "texCoord");
```

Multitexturing

- The ability to use *multiple* textures simultaneously in a shader
- Useful for bump mapping, displacement mapping, etc.
- The different texture units are denoted by GL_TEXTURE0, GL_TEXTURE1, GL_TEXTURE2, etc.
- In simple applications (our homework), we only need one unit

```
void setTextureUnit(GLint unit)
{
  glActiveTexture(unit); // select texture unit affected by subsequent texture calls
  // get a handle to the "textureImage" shader variable
  GLint h_textureImage = glGetUniformLocation(program, "textureImage");
  // deem the shader variable "textureImage" to read from texture unit "unit"
  glUniform1i(h_textureImage, unit - GL_TEXTURE0);
}
```

The display function

```
void display()
{
   // put all the usual code here (clear screen, set up camera, upload
   //   the modelview matrix and projection matrix to GPU, etc.)
   // …

  // select the active texture unit
  setTextureUnit(GL_TEXTURE0); // it is safe to always use GL_TEXTURE0
  // select the texture to use ("texHandle" was generated by glGenTextures)
  glBindTexture(GL_TEXTURE_2D, texHandle);

  // here, bind the VAO and render the object using the VAO (as usual)
  // …

  glutSwapBuffers();
}
```

How are UV maps computed?

- Important subfield of computer graphics!
- Some slides from a past academic talk I have done on this topic.

What is a Good Parametrization?

Different goals for different applications

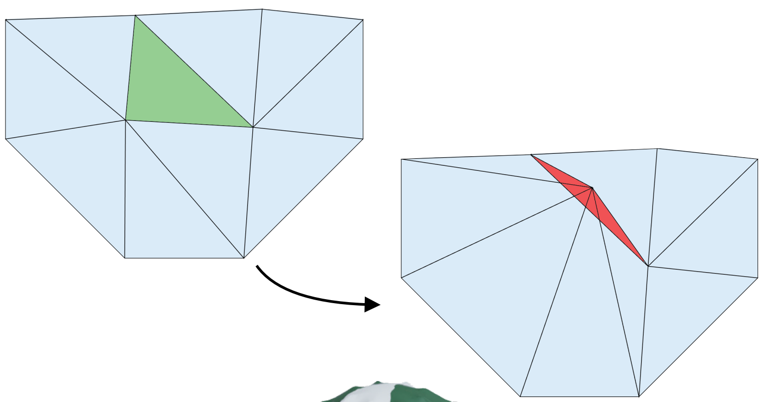
Today our goals are:

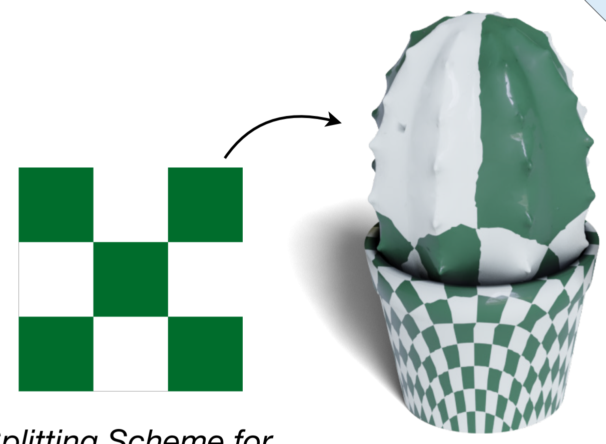
- Flip-free maps
- Distortion-minimizing maps

Useful for UV parametrization

Very narrow path through rich history of research

See article for more complete survey

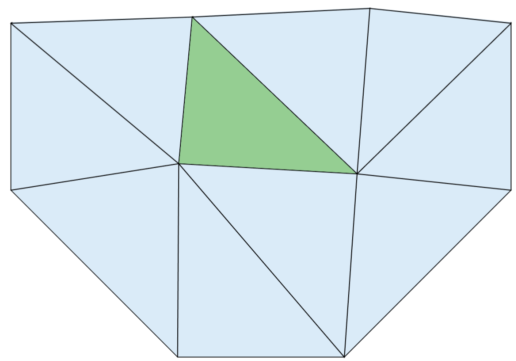




Stein et al. 2022. A Splitting Scheme for Flip-Free Distortion Energies. SIIMS

Flipped Triangles

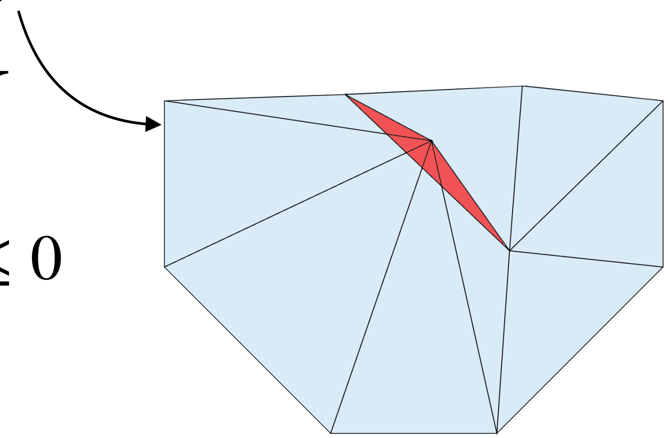
What exactly is a triangle flip?



red triangle mapped to green, flipping it.

Linear map J

$\det J \leq 0$

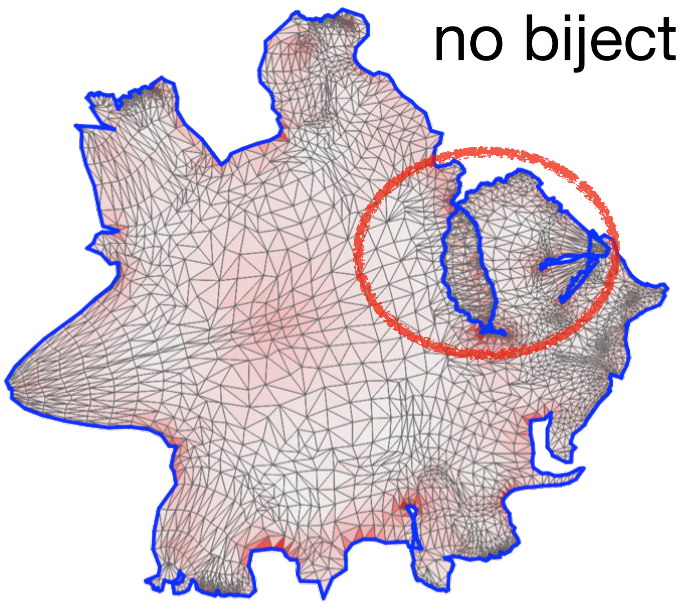


can not separately paint parts of tree

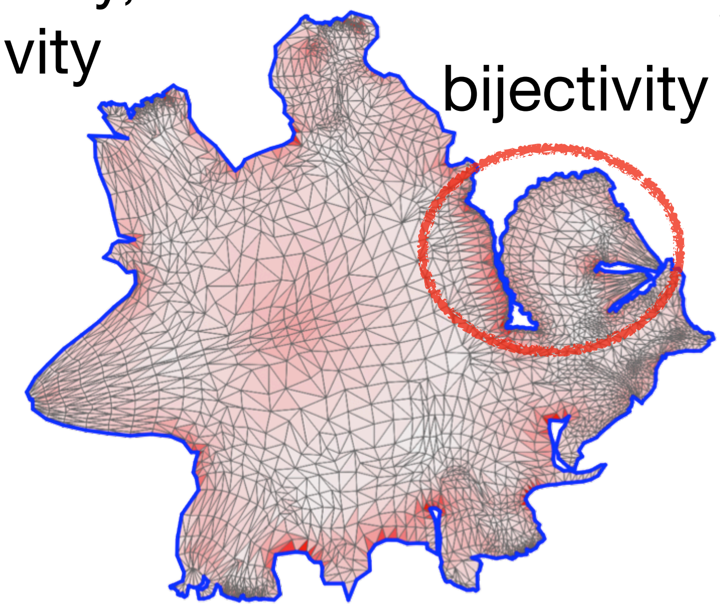
Other Barriers to Bijectivity

Local injectivity vs. global injectivity

local injectivity,
no bijectivity



bijectivity



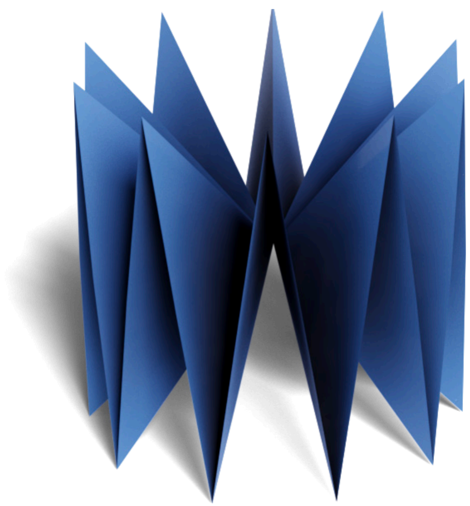
Smith & Schaefer 2015. "Bijective Parameterization With Free Boundaries" (previous work)

Other Barriers to Bijectivity

Stein et al. 2022. A Splitting Scheme for Flip-Free Distortion Energies. SIIMS

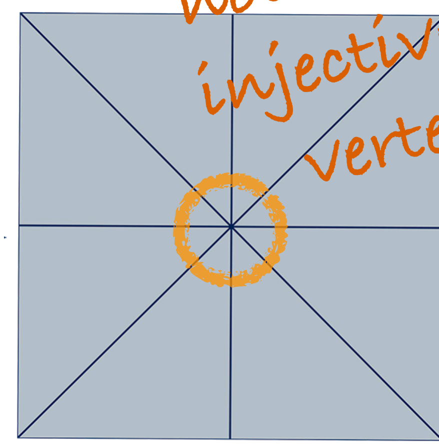
Local injectivity vs. global injectivity

Local injectivity vs. triangle flips

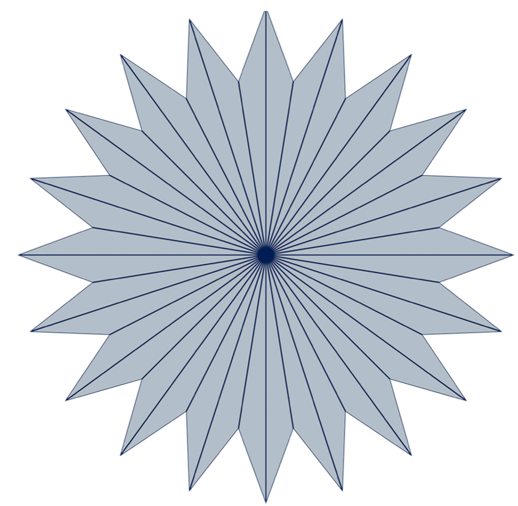


to parametrize

not locally injective at vertex



no triangle flipped
n-cover



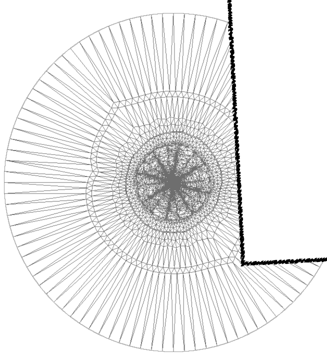
locally injective

Distortion

A large triangle could get mapped to a tiny triangle.

pattern mapped to surface

Larger distortion requires higher-resolution texture.



(this is the flattened cactus)

large, image resolution too low to display anything

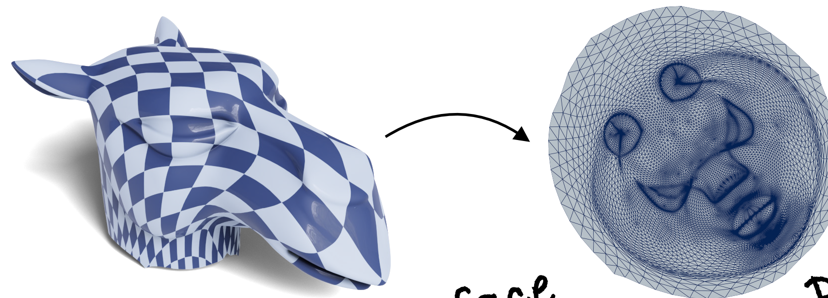
large less

Stein et al. 2022. A Splitting Scheme for Flip-Free Distortion Energies. SIIMS

How to Compute Parametrizations

We want:

- No flipped triangles
- Minimize distortion



Stein et al. 2022. A Splitting Scheme for Flip-Free Distortion Energies. SIIMS

surface plane

parametrization is a map $u : \Omega \to \mathbb{R}^2$

Here: focus on one particular approach (there are many others)

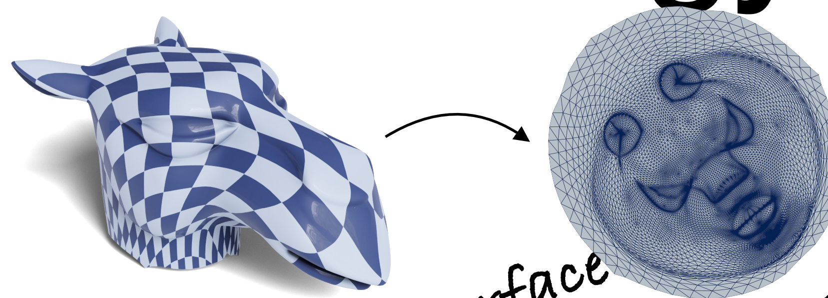
Popular approach: minimize distortion energy depending on the Jacobian D_f

f is linear per triangle/tet $\Rightarrow$ D_f is a constant triangle per triangle/tet

Distortion Energy

We want:

- No flipped triangles
- Minimize distortion



Stein et al. 2022. A Splitting Scheme for Flip-Free Distortion Energies. SIIMS

surface

plane

parametrization is a map $u : \Omega \to \mathbb{R}^2$

Generic distortion energy:

$$E(u) = \sum_{t \in T} w_t f(J_t)$$

t tets/triangles

f defining function

w_t per-element weight (area/volume)

J_t Jacobian of map u on t

Summary

- Introduction
- Filtering and Mipmaps
- Non-color texture maps
- Texture mapping in OpenGL