Zeyang Li
Carnegie Mellon University

Advanced Texturing / GPU Programming

Overview

- Recap: Texture Mapping
- Programmable Graphics Pipeline
- Bump Mapping
- Displacement Mapping
- Environment Mapping
- GLSL Overview
- Perlin Noise
- GPGPU

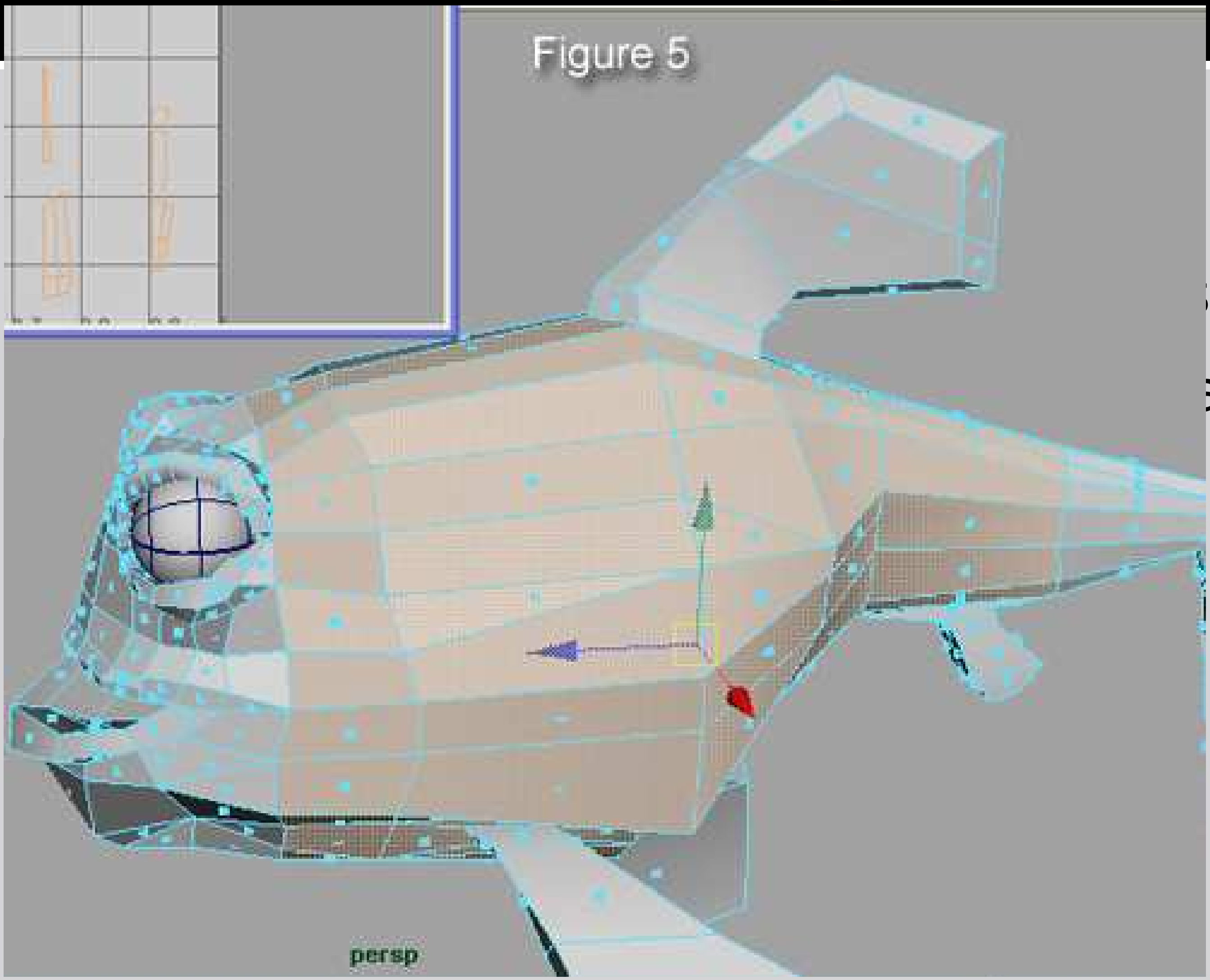
Texture Mapping

- Map reflectance over a piece of geometry
- 2D texture mapping steps:
  - f(x, y, z) mapping function: 3D points to u, v coordinates
  - g(u, v) sampling function: u, v coordinates to color.

Texture Mapping

- The mapping function
  - Easy for simple geometries: cubes, spheres…
  - Not so easy for human body, plant, alien…
    - So it's usually done manually

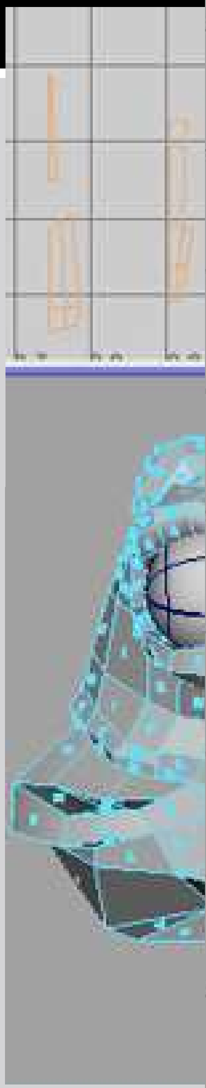
Texture Mapping

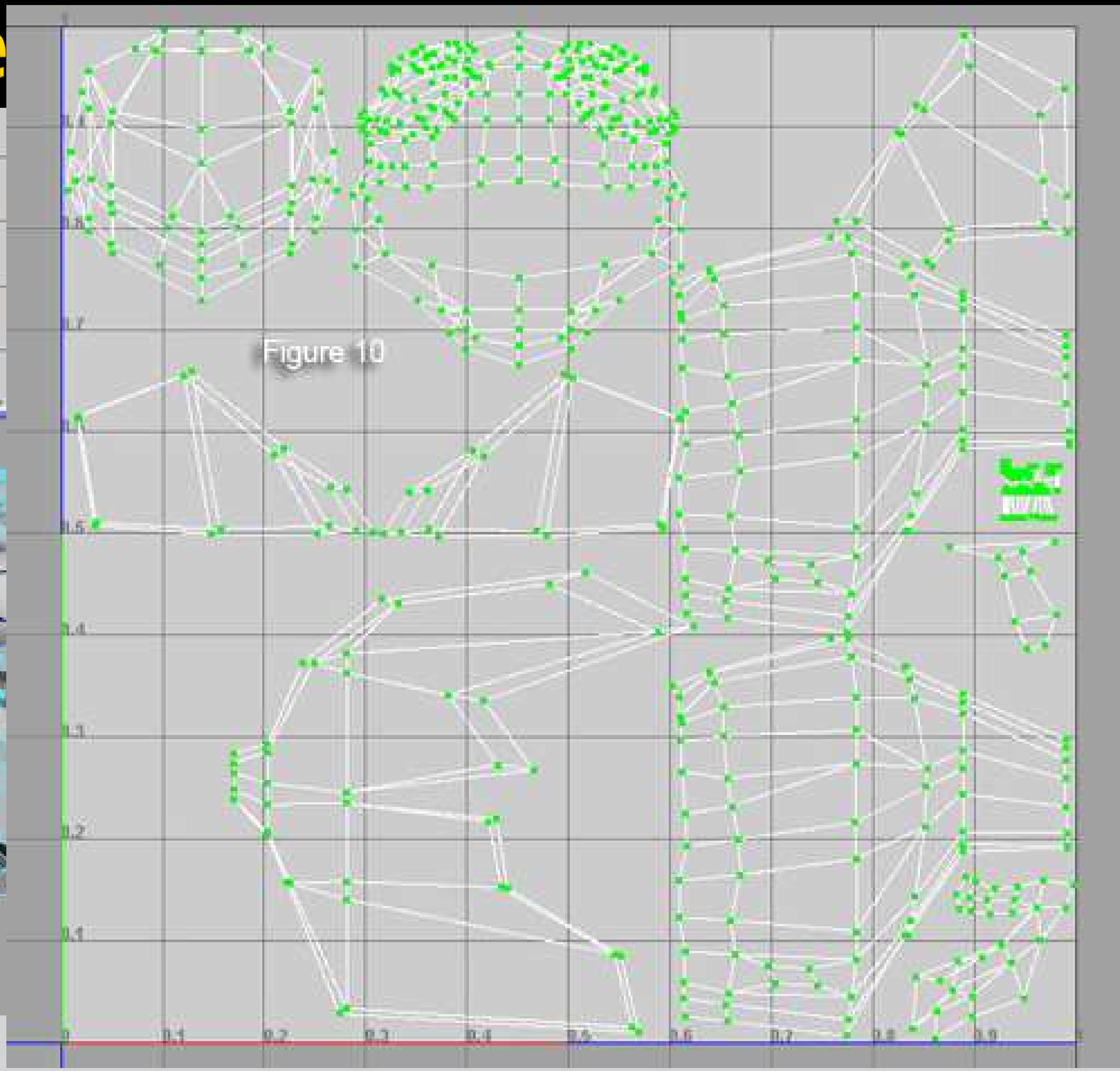


spheres…

alien…

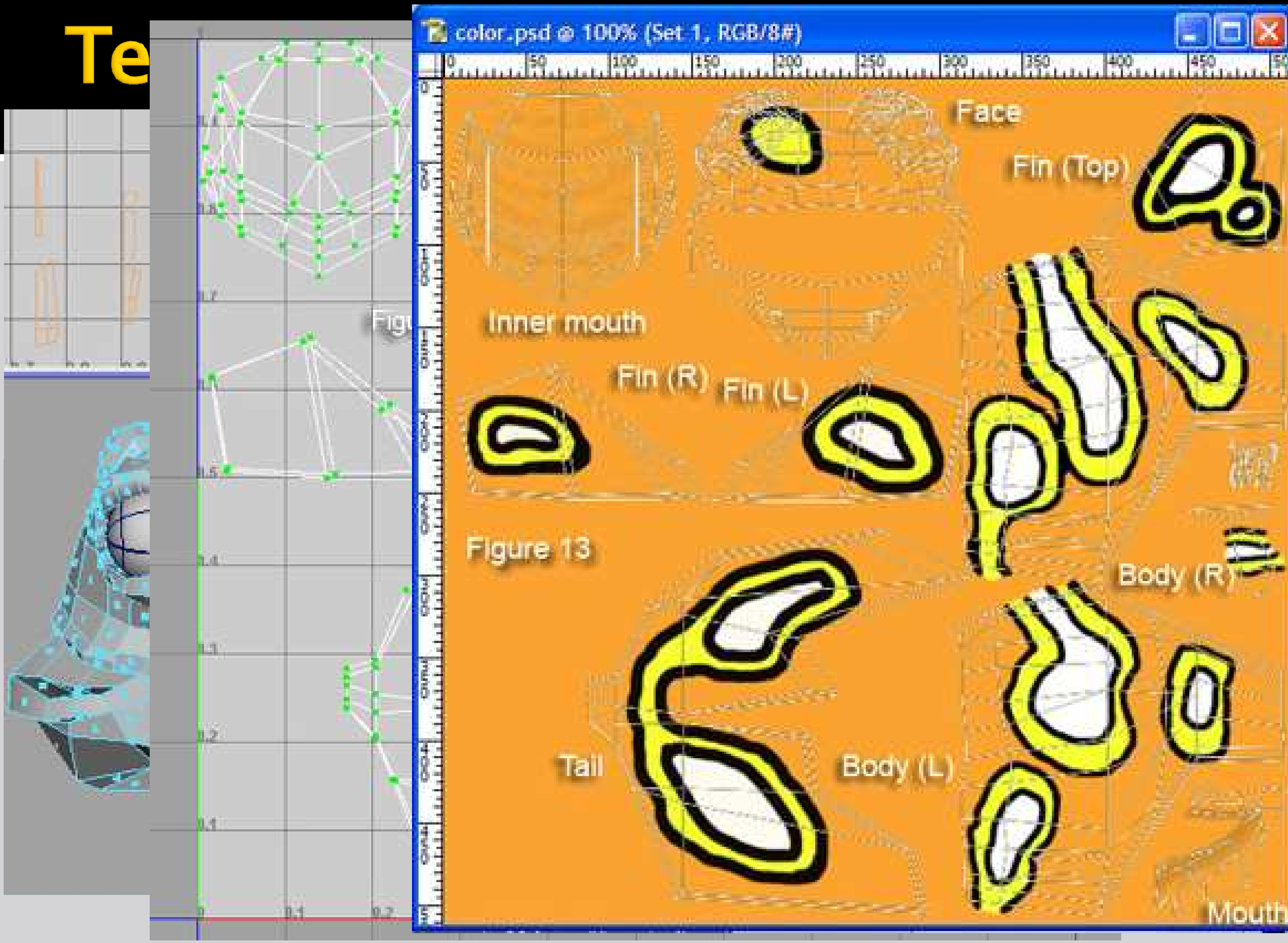
Te



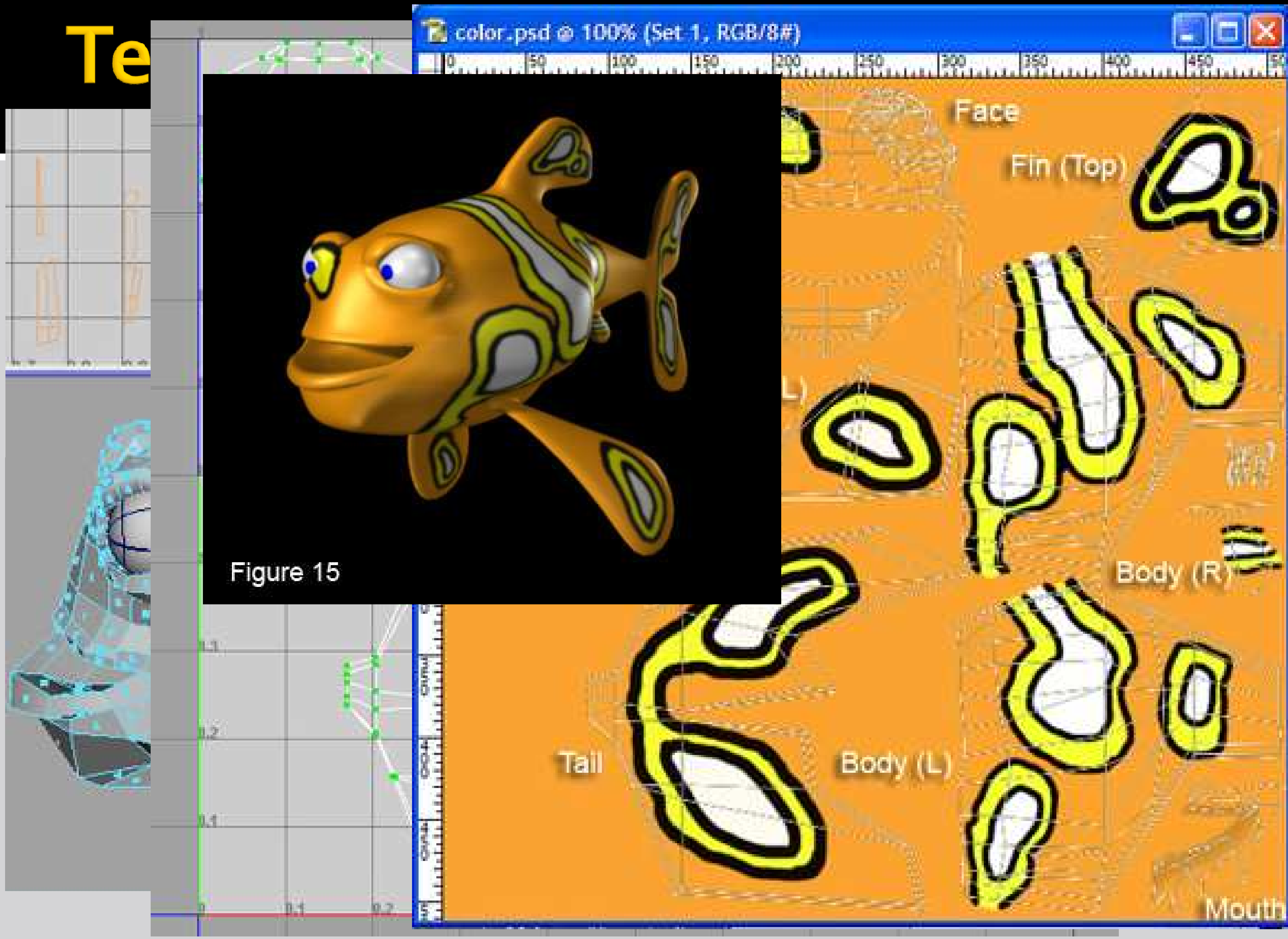


s…

Te


color.psd @ 100% (Set 1, RGB/8#)
Face
Fin (Top)
Inner mouth
Fin (R)
Fin (L)
Figure 13
Body (R)
Tail
Body (L)
Mouth

Te


color.psd @ 100% (Set 1, RGB/8#)
Face
Fin (Top)
Body (R)
Tail
Body (L)
Mouth
Figure 15

Texture Mapping

- The mapping function
  - Easy for simple geometries: cubes, spheres…
  - Not so easy for human body, plant, alien…
    - So it's usually done manually
- You will texture map spheres in project 2(P247)

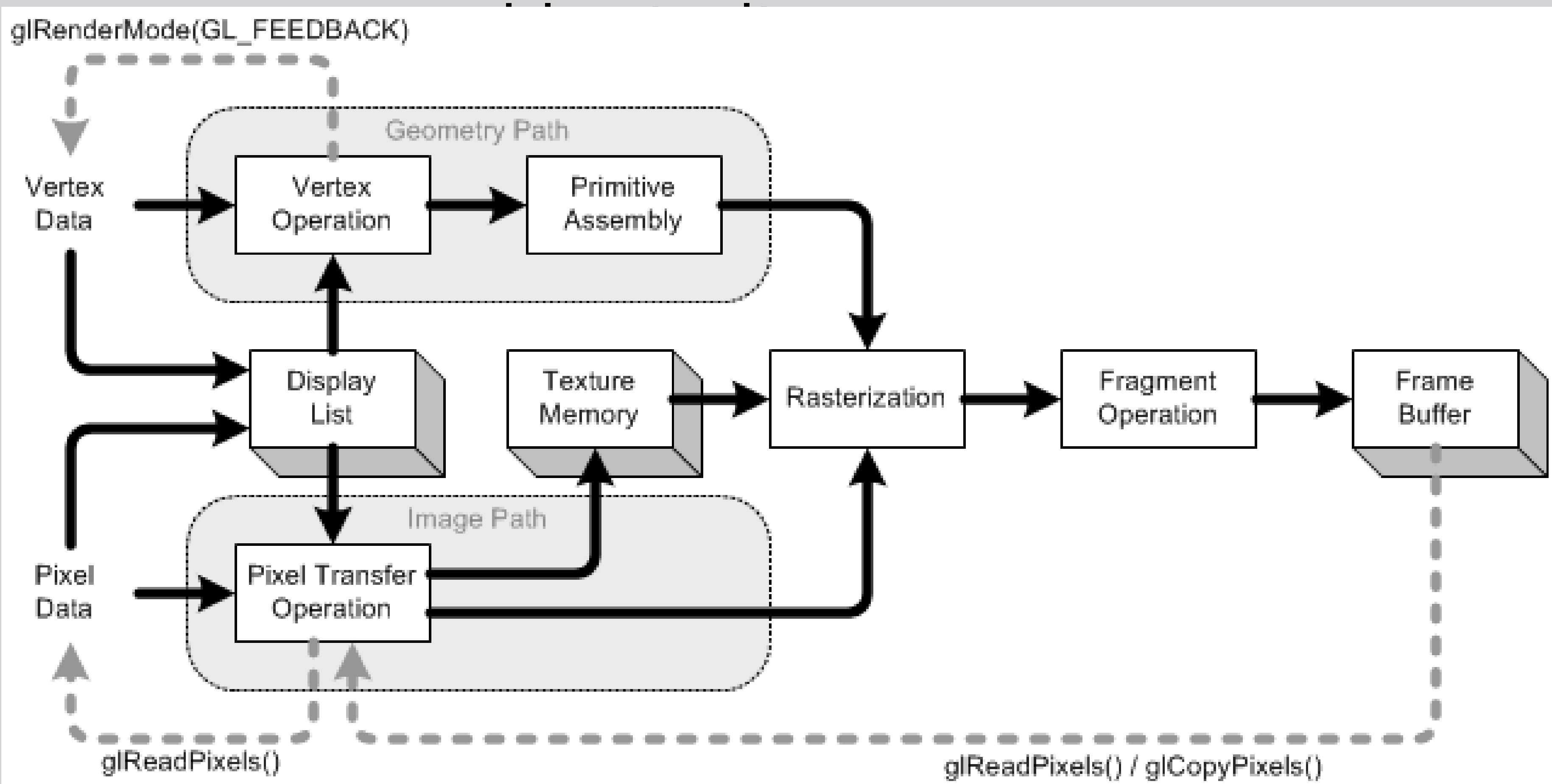
Texture Mapping

- The sampling function (P242)
  - Nearest-neighbor
  - Bilinear
    - Linear interpolation on two directions
  - Hermite
    - Similar to bilinear interpolation, weighting neighbor points differently.

Programmable Graphics Pipeline

- Programmable Pipeline
  - Vertex processors
  - Fragment processors

Programmable Graphics Pipeline



Programmable Graphics Pipeline

- Vertex shader
  - operates on incoming vertices and associated data(normals, uv coordinates).
  - operates on one vertex at a time
  - replaces vertex program in the pipeline
  - must compute the vertex position

Programmable Graphics Pipeline

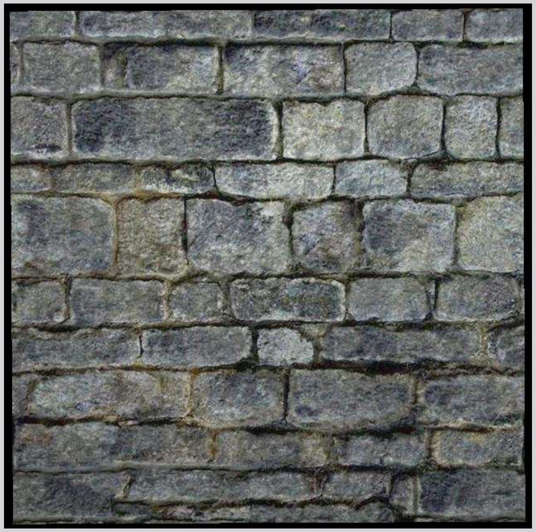
- Fragment shader/pixel shader
  - operates on each fragment
  - fragment: smallest unit being shaded
  - replaces pixel program in the pipeline
  - must compute a color

Programmable Graphics Pipeline

- Programmable Pipeline
  - Vertex processors
  - Fragment processors
- What you can do with it?
  - Anything you can do with fixed function pipeline
  - And a few million more…

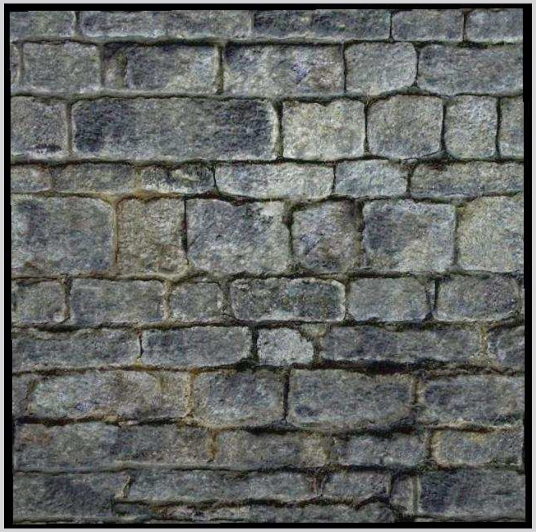
Bump Mapping

- Texture mapping by itself does not produce very satisfying result.
- What can we do to fix it?



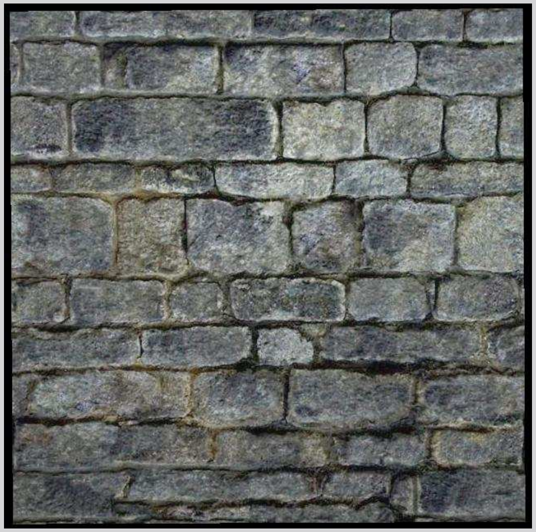
Bump Mapping

- Texture mapping by itself does not produce very satisfying result.
- What can we do to fix it?
  - Normal mapping



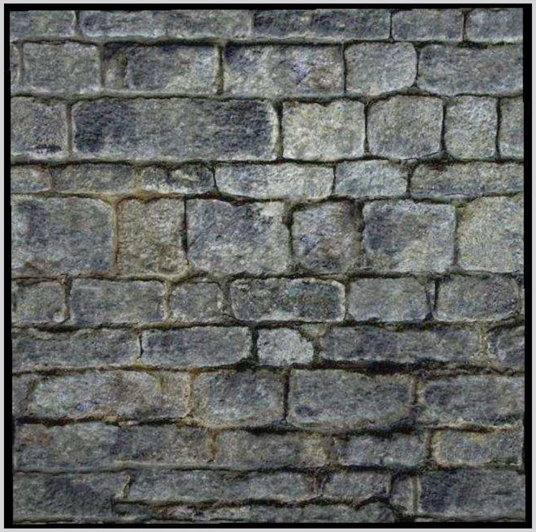
Bump Mapping

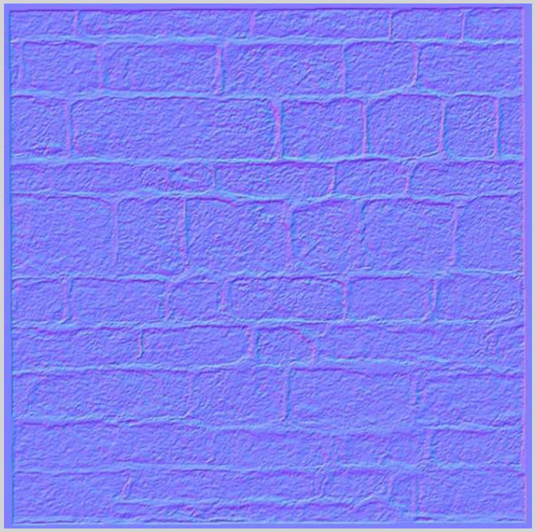
- Texture mapping by itself does not produce very satisfying result.
- What can we do to fix it?
  - perturb the normals



Bump Mapping

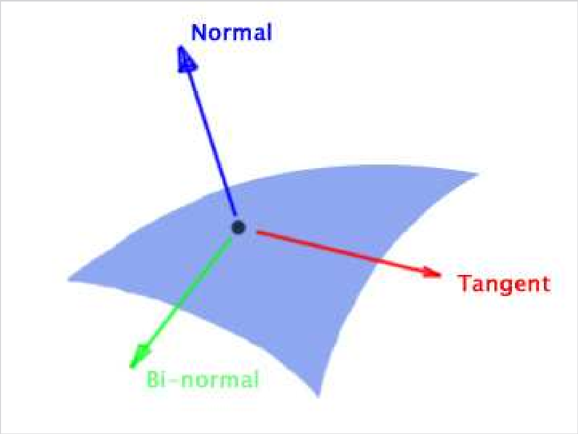
- How?
  - Two textures, color map and normal map





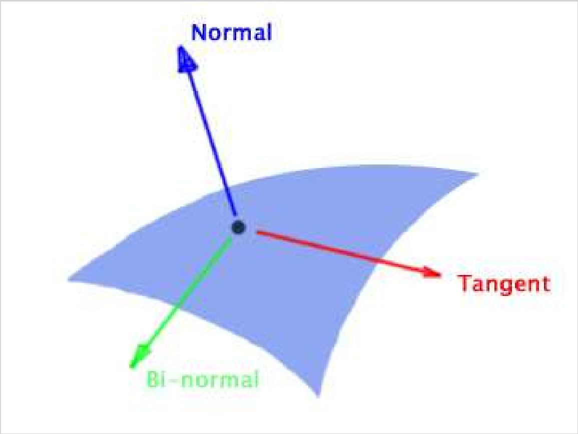
Bump Mapping

- How?
  - Two textures, color map and normal map
  - Normal map usually uses tangent space, while other vectors are in eye space



Bump Mapping

- How?
  - Two textures, color map and normal map
  - Normal map usually uses tangent space, while other vectors are in eye space
  - Eye space to tangent space transformation



Bump Mapping

- Eye space to tangent space transformation

Goal: find basis vectors for tangent space.

We need vertices v1,v2,v3 on a plane, and their (u,v) coordinates c1,c2,c3.

v2v1: 3D(x,y,z) vector from v1 to v2.

c2c1: 2D(u,v) vector from c1 to c2.

Write v2v1 and v3v1 as a linearly combination of the basis vectors T and B.

v2v1 = c2c1.u * T + c2c1.v * B

v3v1 = c3c1.u * T + c3c1.v * B

Solve this linear system, we can get T and B.

N is trivial to compute from T and B. I'm sure you can construct a matrix from T, B and N to transform vectors from eye space to tangent space.

Displacement Mapping

- Bump mapping is not good enough
  - Bumps do not cast shadow or affect sihouette(they don't officially exist…)
- The hard way to do it, add more geometric details.
  - heightmap: displacement in the direction of normals.
  - vertex displacement – possible in vertex shader
  - subvertex displacement
    - Shader model 4.0(DirectX 10), supported only on epic graphics cards(geforce 8800 and above)
    - Requires subdivison, need to generate new vertices

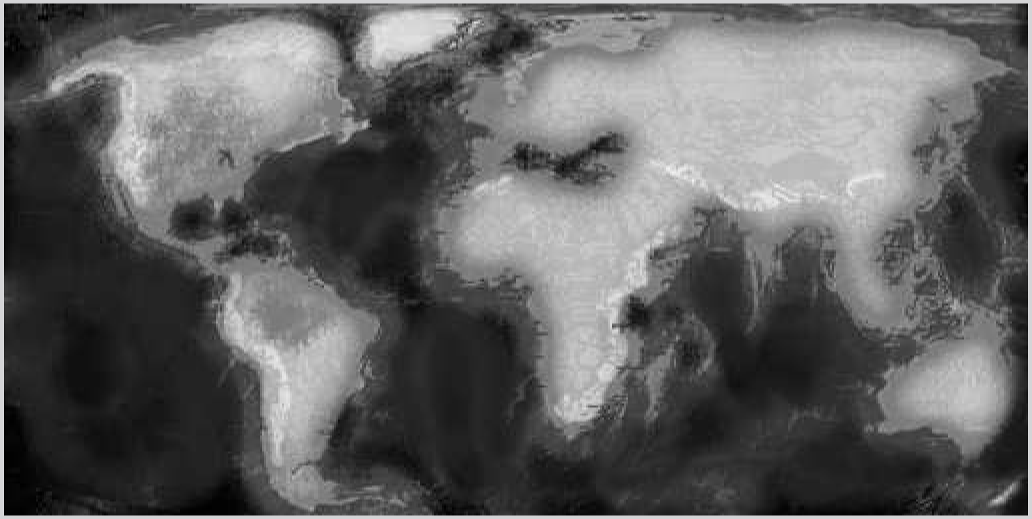
Displacement Mapping

- How?
  - p′ = p + f(p)*n

    f(p): height value from height map

    p: point position

    n: normal



Environment Mapping

- Sometimes Phong shading is not good enough
  - A spaceship traveling in some exotic star system, light sources include 2 suns, 5 planets and a million stars.
  - How many lights we need?

Environment Mapping

- Sometimes Phong shading is not good enough
  - A spaceship traveling in some exotic star system, light sources include 2 suns, 5 planets and a million stars.
  - How many lights we need?
- Environment map is a texture used as “lights”.
  - Good when lighting conditions are static, ex. all light sources are very far away

Environment Mapping

- Basic Idea
  - Convert reflected eye vector to uv coordinates(again, 3D -> 2D, except this time, for real)
  - Different mapping schemes, depending on what environment map you use.

Environment Mapping

- Basic Idea
  - Convert reflected eye vector to uv coordinates(again, 3D -> 2D, except this time, for real)
  - Different mapping schemes, depending on what environment map you use.
- Sphere mapping
  - m = 2*sqrt(x^2 + y^2 + (z+1)^2)

    u = x/m + 0.5

    v = y/m + 0.5

GLSL Overview

- C-like shading language

```
// vertex shader
void main(void)
{
    float shift = 2.0;
    gl_Position = gl_ModelViewProjectionMatrix*gl_Vertex*shift;
}
```

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

GLSL Overview

- C-like shading language

```
// vertex shader
void main(void)
{
    float shift = 2.0;
    gl_Position = gl_ModelViewProjectionMatrix*gl_Vertex*shift;
}
```

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

- More primitives: vec[2-4], mat[2-4], ……
- special types of variables: uniform, varying, attribute

GLSL Overview

- In GLSL, you can access OpenGL states: lighting, materials, modelview matrix, projection matrix, textures, vertex data, ext.
- Vertex shader cannot access texture, or has to suffer a performance penalty. Neither shaders can generate new vertices. Fragment shader cannot change its screen coordinates.

GLSL Overview

- Uniform Variables
  - Read-only
  - Accessible in both shaders
  - Initialized externally
  - GLint diffuse_loc = glGetUniformLocationARB(program, "diffuse");
  - glUniform3fARB(diffuse_loc, 1.0, 0.0, 0.0);

```
uniform vec3 diffuse;
varying vec3 normal;
attribute vec3 tangent;
…

void main(void)
{
  …
}
```

GLSL Overview

- Varying Variables
  - Interface between vertex fragment shader
  - Interpolated automatically
  - Read/write in vertex shader
  - Read-only in fragment shader

```
uniform vec3 diffuse;
varying vec3 normal;
attribute vec3 tangent;
…

void main(void)
{
  …
}
```

GLSL Overview

- Attributes
  - Values passed on a per-vertex base, like position, normal, uv
  - Read-only
  - GLint tangent_loc = glGetAttribLocationARB(program, "tangent");
  - For each vertex, glVertexAttrib3fARB(tangent_loc, t.x, t.y, t.z);

```
uniform vec3 diffuse;
varying vec3 normal;
attribute vec3 tangent;
…

void main(void)
{
  …
}
```

GLSL Overview

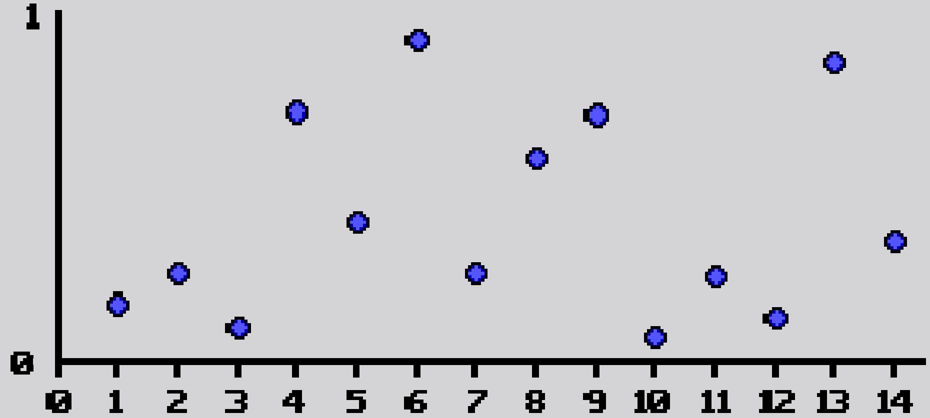
- Demo – Vertex shading vs. Pixel shading
- Useful references
  - GLSL Quick Reference: http://www.opengl.org/sdk/libs/OpenSceneGraph/glsl_quickref.pdf
  - GLSL Specification http://www.opengl.org/registry/doc/GLSLangSpec.Full.1.20.8.pdf

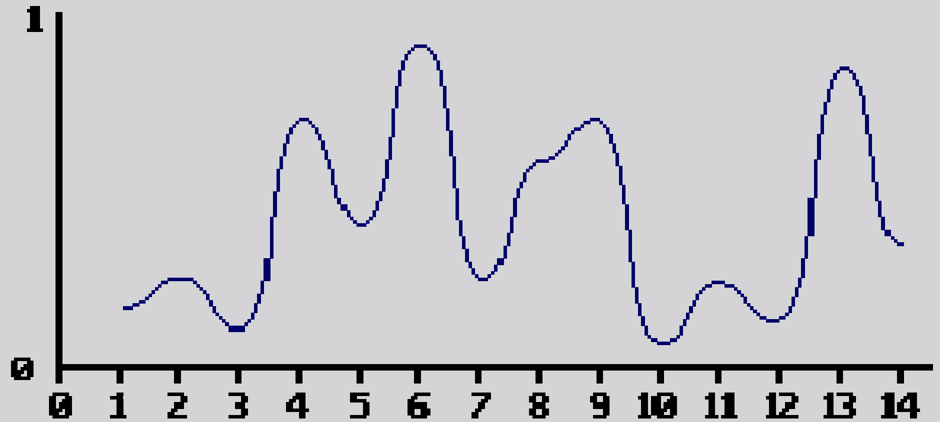
Perlin Noise

- Invented by Ken Perlin
- Functions that generate noise/randomness
- Very useful when generate procedural textures
- Steps to noise
  - Generate random numbers over a KD grid.
  - Interpolate those numbers.
  - Repeat this process at different scales, and add the results together.

Perlin Noise

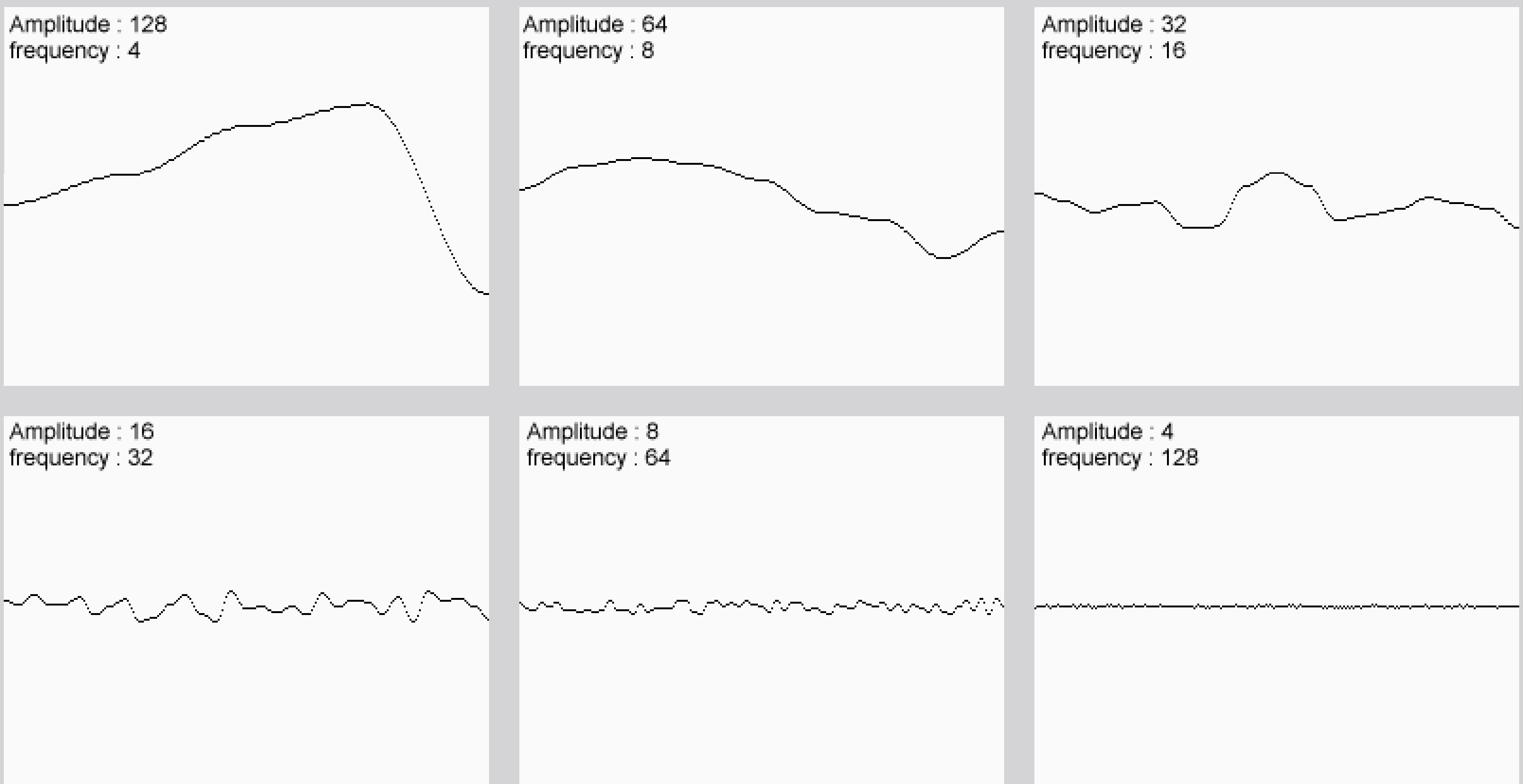
- 1D example





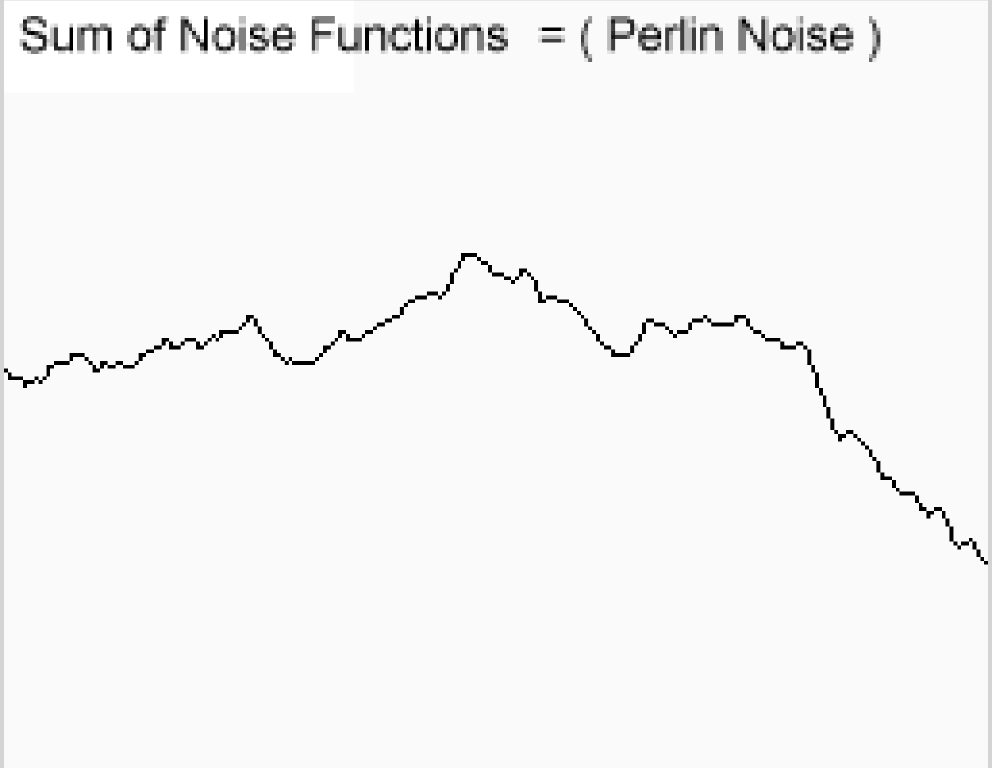
Perlin Noise

- Frequency: density of samples.
  Amplitude, range of samples



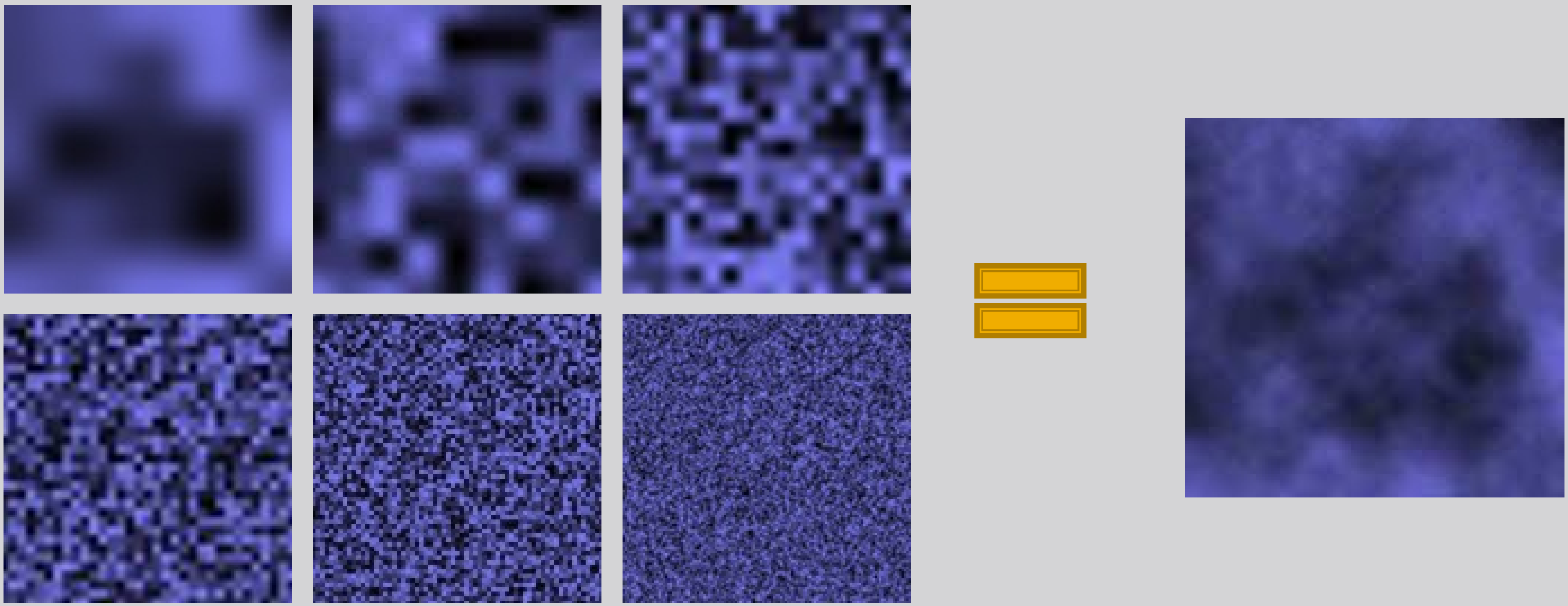
Perlin Noise

- Composition of all the noise



Perlin Noise

- 2D Example



Perlin Noise

- Can do a lot more: cloud, marble, wood, lava, gold…
- Interpolation
  - Linear, cosine, cubic, Gaussian blur…
- A more comprehensive treatment of Perlin noise
  - P69, Texturing & Modeling: A Procedural Approach 3rd
  - http://freespace.virgin.net/hugo.elias/models/m_perlin.htm

GPGPU

- General-Purpose compute on GPU (GPGPU)
- Much faster than CPU for highly parallelized computation.
- Hard to program
  - have encode data as vertices/textures, feedback through write-to-texture
- New platforms are coming out that support more CPU-like programming style, Intel's Larrabee, nVidia's CUDA.

Reference

- The textbook
- About Perlin noise: http://freespace.virgin.net/hugo.elias/models/m_perlin.htm
- GPGPU slides: http://www.gpgpu.org/s2007/slides/01-introduction.pdf