Texture Transfer
Texture Transfer

OBJ:

v 0.1 1 0.5
v 1 1.1 0.5
v 0.3 0 0.8
vt 0.4 0.3
vt 0.3 0.8
vt 0.85 0.5
f 1/1 2/2 3/3
Texture Transfer
Texture Transfer

• Computing texture coordinates (parameterization to the plane) is not trivial
  – There exist simple methods, such as:
    • Projection on a plane
    • Map to a sphere, use latitude and longitude

• Anti aliasing techniques are useful if the screen resolution is too low
  – Mipmap: adjust image resolution to the screen resolution
Textures in OpenGL

- Vertex shader:

```glsl
in vec2 texcoord;
in vec4 vPosition;
...
out vec2 st;
...
void main()
{
...
    gl_Position=vPosition;
    st=texcoord;
}
```
Textures in OpenGL

- **Fragment shader:**

```cpp
in vec2 st;
uniform sampler2D texMap
out vec4 color;
void main()
{
    color = texture2D(texMap, st);
}
```
Textures in OpenGL

• CPP:

```cpp
// bind texture i and the image texels
glActiveTexture(GL_TEXTURE_i);
GLuint tex;
glGenTextures(1, &tex);
glBindTexture(GL_TEXTURE2D, tex)
GLubyte texels[512][512][3];
texels = ...
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, 512,
    512, 0, GL_RGB, GL_UNSIGNED_BYTE, texels);

tex_loc = glGetUniformLocation(program,"texMap");
glUniform1i(tex_loc, i);
```
Environment Mapping
Bump Mapping

• Change the normal that is used for lighting:

\[ I = I_d k_a + I_p \left( k_d (N \cdot L) + k_s (R_N \cdot V)^n \right) \]
Normal Mapping

RGB can be used as xyz values that specify normals in object frame.
Displacement Mapping

- Adjusting normals is not enough, adjust vertex positions as well:
Displacement Mapping

• Adjusting normals is not enough, adjust vertex positions as well:

• If a height map is given, move each vertex in the normal direction according to the height
Displacement Mapping

- Adjusting normals is not enough, adjust vertex positions as well:

  Normal map
  Normal + Displacement
  Texture Mapped
Displacement Mapping

• If we use rgb values as xyz in object frame, it will only be suitable for specific objects

• Normal map in *tangent space*: rgb are coordinates that are relative to a local frame

• Blue is the coordinate in the normal direction
Tangent and Bitangent

- For each face, compute a tangent vector $T$ that “points to the right of the texture”:

3D vertices
Tangent and Bitangent

- For each face, compute a tangent vector $T$ that “points to the right of the texture”:

  $\begin{align*}
  u & = (0, 1) \\
  v & = (0, 0) \\
  (1, 1) & = (1, 0) \\
  E_1 & \\
  E_2 & \\
  B & \\
  \Delta_1 & \\
  \Delta_2 &
  \end{align*}$
Tangent and Bitangent

- For each face, compute a tangent vector $T$ that “points to the right of the texture”:

\[
\begin{bmatrix}
E^x_1 & E^y_1 & E^z_1 \\
E^x_2 & E^y_2 & E^z_2
\end{bmatrix}
= \begin{bmatrix}
\Delta^u_1 & \Delta^v_1 \\
\Delta^u_2 & \Delta^v_2
\end{bmatrix}
\begin{bmatrix}
T^x & T^y & T^z \\
B^x & B^y & B^z
\end{bmatrix}
\]
Examples

High resolution mesh

Low resolution mesh

displacement-mapped
Procedural Texture Generation

- Compute color at each point
- Bombing: basic shapes in random locations and sizes
**Procedural Texture Generation**

- Compute color at each point
- Useful building block:
  - `marble_color(float x)` gets a float and returns a color, that is a mixture of two colors
  - For example:
    \[
    \frac{\sin(x) + 1}{2} \text{color1} + \left(1 - \frac{\sin(x) + 1}{2}\right) \text{color2}
    \]
Turbulence

• Noise can be useful to generate textures

• Random noise does not look good:

• We need “smooth noise”
Turbulence

- Generate noisy, then apply a smoothing filter
Marble Texture

```plaintext
function marble(p)
    x = p.x + turbulence(p);
    return marble_color(sin(x))
```

function wood(p)
    x = (p.x^2 + p.y^2) + turbulence(point);
    return wood_color(sin(x))