CS 563 Advanced Topics in Computer Graphics

Texture Mapping

by Steve Olivieri
A texture is “a color that varies with location.”

Textures modify the way a material responds to light.
  - Materials – reflection, scattering, transmission, emission, absorption
  - Bump Map, Displacement Map, Clip Map

Declaration:

```cpp
class Texture {
    public:
        // constructors, etc.

        virtual RGBColor get_color(const ShadeRec& sr) const = 0;
};
```
The constant color texture returns a specified color, regardless of location.

Useful for specular highlights, reflection coefficients, etc.

```cpp
class ConstantColor : public Texture {
    public:
        // constructors, etc.

        void set_color(const RGBColor& c);
        virtual RGBColor& get_color(const ShadeRec& sr) const;

    private:
        RGBColor color;
};

RGBColor ConstantColor::get_color(const ShadeRec& sr) const {
    return color;
}
```
The **image texture** returns colors based on an image (e.g. JPEG, TIFF, PPM).

Image textures are a cheap, efficient way to add detail to a surface.

```cpp
class ImageTexture : public Texture {
public:
    // constructors, etc.
    virtual RGBColor& get_color(const ShadeRec& sr) const;

private:
    int hres, vres;
    Image* image_ptr;
    Mapping* mapping_ptr;
};
```
The color returned from `get_color()` is determined by the local hit point \((u, v)\).
- Calculated at ray intersection.
- Stored in ShadeRec.

These coordinates might be displaced by a mapping.

```cpp
RGBColor ImageTexture::get_color(const ShadeRec& sr) const {
    int row, column;

    if(mapping_ptr)
        mapping_ptr->get_texel_coordinates(sr.local_hit_point, hres,
                                            vres, row, column);
    else {
        row = (int)(sr.v * (vres - 1));
        column = (int)(sr.u * (hres - 1));
    }

    return (image_ptr->get_color(row, column));
```
The properties of spatially varying materials (and the spatially varying BRDFs that define them) change with location.

- RGBColor values are replaced with Texture pointers.
- Any $c^*$ or $k^*$ value can be a texture!
class SV_Lambertian : public BRDF {
    public:
        // constructors, etc.
        virtual RGBColor rho(const ShadeRec& sr, ...);
        virtual RGBColor f(const ShadeRec& sr, ...);
        virtual RGBColor sample_f(const ShadeRec& sr, ...);
    private:
        float kd;
        Texture* cd;
};

RGBColor SV_Lambertian::rho(const ShadeRec& sr, ...) {
    return (kd * cd->get_color(sr));
}

RGBColor SV_Lambertian::f(const ShadeRec& sr, ...) {
    return (kd * cd->get_color(sr) * invPI);
}
class SV_Matte : public BRDF {
    public:
        // constructors, etc.
        void set_cd(const Texture* t_ptr);
        virtual RGBColor shade(ShadeRec& sr);
    private:
        SV_Lambertian* ambient_brdf;
        SV_Lambertian* diffuse_brdf;
};

inline void SV_Matte::set_cd(const Texture* t_ptr) {
    ambient_brdf->set_cd(t_ptr);
    diffuse_brdf->set_cd(t_ptr);
}

The shade function does not change!
- Each type of surface requires a different map to translate the 3D hit point into the correct pixel in a 2D texture.
- The 2D texture coordinates are *normalized*, so $(u, v) \in [0,1] \times [0,1]$.
- Converting from $(u, v)$ to texel coordinates $(x_p, y_p)$:
  - $x_p = (h_{res} - 1)u$
  - $y_p = (v_{res} - 1)v$
Rectangular Mapping

- Map to a generic rectangle \((x, z) \in [-1, +1] \times [-1, +1]\).
  - \(u = (z + 1) / 2\)
  - \(v = (x + 1) / 2\)
- This maps the entire texture, regardless of size or aspect ratio.
- Use transformations to adjust the rectangle.
Using cylindrical coordinates, hit point $p$ is defined by $\Phi \in [0, 2\pi)$ and $y \in [-1, +1]$.

- $\Phi = \tan^{-1}(x/z)$
- $u = \Phi/2\pi$
- $v = (y + 1)/2$

Maps the left side of the image onto the line where the generic cylinder intersects the $(y, z)$ plane with $x = 0$ and $z = 1$.

Image must tile horizontally to avoid discontinuity.
- Spheres are not mathematically flat! Use Mercator.
- $\phi$ is the same as the cylindrical map map.
  - $\theta = \cos^{-1}(y)$
  - $u = \phi/2\pi$
  - $v = 1 - \theta/\pi$
- Because $\phi \in [0,2\pi]$ and $\theta \in [0,\pi]$, textures must have a 2:1 aspect ratio to cover the sphere.
Light-Probe Mapping

- Place a reflective sphere in the middle of a scene and photograph it multiple times to sample the whole environment.
- The area directly behind the sphere is not reflected without multiple photographs.
Light-Probe Mapping

- Hit point has coordinates \((x, y, z) \in [-1, +1]^3\) on a unit sphere at the origin. Then,
  - \(\alpha = \cos^{-1}(z)\)
  - \(\sin \beta = y/(x^2 + y^2)^{1/2}\)
  - \(\cos \beta = x/(x^2 + y^2)^{1/2}\)
- So, the \((u, v)\) coordinates are:
  - \(u = [1.0 + (\alpha/\pi) \cos \beta]/2.0\)
  - \(v = [1.0 + (\alpha/\pi) \sin \beta]/2.0\)
Light-Probe Mapping

- Create a texture from the photographs and apply it to a large sphere surrounding the scene.
- Gives the illusion of being inside the photographed environment.
- Light-Probed images are mirror reversals of reality, so flip $z$ for actual panorama photographs.
Light-Probe Mapping
If pixels and texels do not match exactly, image textures will suffer from aliasing.

- A given pixel covers many texels. The resulting texture generally looks good.
- A given pixel covers only a fraction of a texel. The resulting texture looks pixelated.

To prevent aliasing,

- Use textures with high resolutions
- Do not zoom in close on objects with image-based textures
- Use intrinsic antialiasing, where the texture antialiases itself by averaging neighboring pixels for the returned color.
- Triangles meshes are the most commonly textured objects in commercial ray tracers.
- To add support for textures triangle meshes, add two new parameters to the PLY file for \( u \) and \( v \).
- Two new triangle types:
  - FlatUVMeshTriangle
  - SmoothUVMeshTriangle
- The \( u \) and \( v \) values are interpolated in the base MeshTriangle class since the process is the same for both flat- and smooth-shaded triangles.
ply
format ascii 1.0
element vertex 4
property float x
property float y
property float z
property float u
property float v
element face 2
property list int int vertex_indices
end_header
-1.0 0.0 -1.0 0.0 0.0 0.0
-1.0 -1.0 1.0 1.0 0.0 0.0
1.0 0.0 1.0 1.0 1.0 1.0
1.0 -1.0 -1.0 0.0 1.0 1.0
3 0 1 2
3 0 2 3
Example: Earth
Example: Earth
Example: Earth
Example: Two Triangles
Example: Two Triangles
QUESTIONS?