CS 563 Advanced Topics in Computer Graphics Materials

by Paulo Gonçalves de Barros
Summary

- Basic concepts
- BSDF
- Material interface and implementations
- Bump mapping
- Scene structure

Diagram:

```
Scene
  ↓
Primitives
  ↓
  Shape
  ↓
  Material
```
Calculating hits

- Scene
  - Primitives
    - Intersection
      - Shape
    - Material
- Obtaining ray colors

Diagram:

```
Scene -> Primitives
  |
  v
Shape  Material
```

`Basic concepts`
Basic concepts

- Obtaining ray colors
Obtaining ray colors

- BSDF

Whitted’s

<Compute emitted and reflected light at ray intersection point>
<Evaluate BSDF at hit point>
Obtaining ray colors

1. BSDF
2. Area light?

Whitted's

<Compute emitted and reflected light at ray intersection point>
<Evaluate BSDF at hit point>
<Initialize common variables for Whitted integrator>
<Compute emitted light if ray hit an area light source>
- Obtaining ray colors

1. Compute emitted and reflected light at ray intersection point
2. Evaluate BSDF at hit point
3. Initialize common variables for Whitted integrator
4. Compute emitted light if ray hit an area light source
5. Add contribution of each light source

Whitted’s
### Obtaining ray colors

- **1. BSDF**
  - Compute emitted and reflected light at ray intersection point.
  - Evaluate BSDF at hit point.
  - Initialize common variables for Whitted integrator.
  - Compute emitted light if ray hit an area light source.
  - Add contribution of each light source.

- **2. Area light?**

- **3. Light contrib.**

- **4. Reflection and refraction**

---

Whitted's algorithm:

```
if (rayDepth++ < maxDepth) {
    <Trace rays for specular reflection and refraction>
}
--rayDepth;
```
- Obtaining ray colors

**Scene**

**Primitives**

**Shape**

**Material**

**BSDF**

**Texture**

**BRDF**

**BTDF**
- Parameters
  - DifferentialGeometry - shading dg
  - Geometric normal - n_G
  - Index of refraction

- Builds orthonormal coordinate system
  - Shading normal n_S
- Fixed maximum number of BxDFs
  - 8
  - Never needed more than that.
- Methods
  - Number of BxDFs Components
  - Normal equality
  - Coord. frames transformations
Problems with shading normals
- Problems with shading normals
  - Light leak

\[ n_s \quad n_G \]
- Problems with shading normals
  - Light leak
  - Black spots
Solution

- BRDF or BTDF?
  - Use geometric normal;

Evaluate BRDFs

Evaluate BTDFs
- **Solution**
  - BRDF or BTDF?
    - Use geometric normal;

  ![Diagram showing BRDF evaluation](Diagram)

  ![Diagram showing BTDF evaluation](Diagram)

- **Scattering equation evaluation**
  - Use shading normal
- Solution

- Light leaks avoided
- Only BTDFs are considered

- Black spots avoided
- Only BRDFs are considered
- Memory management
  - Many BSDFs created during single ray cast
  - Performance issues with dynamic allocation
- Memory management
  - Many BSDFs created during single ray cast
  - Performance issues with dynamic allocation

- How to avoid this?
Memmory management

- Many BSDFs created during single ray cast
- Performance issues with dynamic allocation

- How to avoid this?
- Previous memory allocation
- Memory management
  - Many BSDFs created during single ray cast
  - Performance issues with dynamic allocation

- How to avoid this?
  - Previous memory allocation

- For every single ray hit?
Memory management

- Many BSDFs created during single ray cast
- Performance issues with dynamic allocation

How to avoid this?

- Previous memory allocation

For every single ray hit?

- Reuse memory
- Memory Arena
  - Static chunk of memory
  - All BxDFs for a ray are sequentially saved there
  - Used and recycled at every ray tracing

```cpp
Class BSDF {
    static Memory Arena
    .
    .
    .
}
```
GetBSDF method

Parameters
- $dgGeom$ – actual Differential Geometry
- $dgShading$ – perturbed shading geometry

Returns final shading geometry for point
- BRDF
- BTDF
Create access to BSDF in Intersection class

- Intersection:: GetBSDF
  - dg.ComputeDifferentials (ray)
  - Primitive->getBSDF
  - Material->getBSDF
Materials

- Matte
  - Purely diffuse surface
  - Parameters
    - Spectral diffuse reflection - Kd
    - Scalar roughness value - sigma
    - Optional scalar texture - bumpMap
Materials

- Sigma variation example

Sigma = 0 ? Lambertian BRDF
- Sigma variation example

\[ \Sigma = 1 \]

OrenNayar BRDF
- Matte getBSDF method

BSDF *Matte::GetBSDF(const DifferentialGeometry &dgGeom,
const DifferentialGeometry &dgShading) const {
    // Allocate _BSDF_, possibly doing bump-mapping with _bumpMap_
    DifferentialGeometry dgs;

    return bsdf;
}
### Matte getBSDF method

BSDF *Matte::GetBSDF(const DifferentialGeometry &dgGeom, const DifferentialGeometry &dgShading) const {
  // Allocate _BSDF_, possibly doing bump-mapping with _bumpMap_
  DifferentialGeometry dgs;

  if (bumpMap)
    Bump(bumpMap, dgGeom, dgShading, &dgs);
  else
    dgs = dgShading;

  return bsdf;
}
Matte getBSDF method

BSDF *Matte::GetBSDF(const DifferentialGeometry &dgGeom,
                    const DifferentialGeometry &dgShading) const {
    // Allocate _BSDF_, possibly doing bump-mapping with _bumpMap_
    DifferentialGeometry dgs;

    if (bumpMap)
        Bump(bumpMap, dgGeom, dgShading, &dgs);
    else
        dgs = dgShading;

    BSDF *bsdf = BSDF_ALLOC(BSDF)(dgs, dgGeom.nn);
    return bsdf;
}
**Matte getBSDF method**

```cpp
BSDF *Matte::GetBSDF(const DifferentialGeometry &dgGeom, const DifferentialGeometry &dgShading) const {
    // Allocate _BSDF_, possibly doing bump-mapping with _bumpMap_
    DifferentialGeometry dgs;

    if (bumpMap)
        Bump(bumpMap, dgGeom, dgShading, &dgs);
    else
        dgs = dgShading;

    BSDF *bsdf = BSDF_ALLOC(BSDF)(dgs, dgGeom.nn); // Allocates the BSDF

    Spectrum r = Kd->Evaluate(dgs).Clamp(); // Texture evaluation; Obtention of reflection and roughness coefficients.

    return bsdf;
}
```

- Calculates shading normal based on bump map
- Allocates the BSDF
- Texture evaluation; Obtention of reflection and roughness coefficients.
Matte getBSDF method

BSDF *Matte::GetBSDF(const DifferentialGeometry &dgGeom,
     const DifferentialGeometry &dgShading) const {
    // Allocate _BSDF_, possibly doing bump-mapping with _bumpMap_
    DifferentialGeometry dgs;

    if (bumpMap)
        Bump(bumpMap, dgGeom, dgShading, &dgs);
    else
        dgs = dgShading;

    BSDF *bsdf = BSDF_ALLOC(BSDF)(dgs, dgGeom.nn); // Allocates the BSDF

    Spectrum r = Kd->Evaluate(dgs).Clamp(); // Texture evaluation; Obtention of
    // reflection and roughness coefficients.

    float sig = Clamp(sigma->Evaluate(dgs), 0.f, 90.f);
    if (sig == 0.)
        bsdf->Add(BSDF_ALLOC(Lambertian)(r));
    else
        bsdf->Add(BSDF_ALLOC(OrenNayar)(r, sig));

    return bsdf;
}
- Plastic
  - Mixture of diffuse and glossy surface
  - Parameters
    - Spectral diffuse reflection – $K_d$
    - Glossy specular reflection – $K_s$
    - Scalar roughness value – roughness
      - Size of specular highlight
    - Optional scalar texture – bumpMap
Plastic getBSDF method

BSDF *Plastic::GetBSDF(const DifferentialGeometry &dgGeom,
const DifferentialGeometry &dgShading) const {
    DifferentialGeometry dgs;
    if (bumpMap) {
        Bump(bumpMap, dgGeom, dgShading, &dgs);
    } else {
        dgs = dgShading;
    }

    BSDF *bsdf = BSDF_ALLOC(BSDF)(dgs, dgGeom.nn);

    Spectrum kd = Kd->Evaluate(dgs).Clamp();  \{ Texture reflection evaluation; \}
    BxDF *diff = BSDF_ALLOC(Lambertian)(kd);
    Fresnel *fresnel = BSDF_ALLOC(FresnelDielectric)(1.5f, 1.f);
    bsdf->Add(diff);

    Spectrum ks = Ks->Evaluate(dgs).Clamp();  \{ Texture glossy evaluation; \}
    float rough = roughness->Evaluate(dgs);
    BxDF *spec = BSDF_ALLOC(Microfacet)(ks, fresnel,
                 BSDF_ALLOC(Blinn)(1.f / rough));
    bsdf->Add(spec);

    return bsdf;
}
Materials

Plastic
Other materials
- Translucent
- Mirror
- Glass
- ShinyMetal
- Substrate
- Clay
- Felt
- Primer
- Skin
- BluePaint
- Uber
Blue paint
Materials

Glass
Bump Mapping

- Displacement simulation to points

\[ p'(u, v) = p(u, v) + d(u, v)n(u, v) \]
Bump Mapping

- Displacement simulation to points

\[ p'(u, v) = p(u, v) + d(u, v)n(u, v) \]
Bump Mapping

- Displacement simulation to points

\[ p'(u, v) = p(u, v) + d(u, v)n(u, v) \]
Bump Mapping

\[ n = \frac{\partial p}{\partial u} \times \frac{\partial p}{\partial v} \]

- Change the partial derivatives of \( p \) to change normal for \( p' \)
  \[ p'(u, v) = p(u, v) + d(u, v)n(u, v) \]

- Derivation using the product and chain rules
  \[ \frac{\partial p'}{\partial u} = \frac{\partial p}{\partial u} + \frac{\partial d}{\partial u} n + d \frac{\partial n}{\partial u} \]

<table>
<thead>
<tr>
<th>Product rule</th>
<th>Chain rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>( D[u<em>v] = u</em>D[v] + v*D[u] )</td>
<td>( h(t) = f(g(t)) ).</td>
</tr>
<tr>
<td></td>
<td>( dh(t) = df(g(t))dg(t) ).</td>
</tr>
</tbody>
</table>
By the definition of partial derivative
\[
\frac{\partial d(u, v)}{\partial u} = \lim_{\Delta u \to 0} \frac{d(u + \Delta u, v) - d(u, v)}{\Delta u}
\]

For small \(\Delta u\), we have that
\[
\frac{\partial p'}{\partial u} = \frac{\partial p}{\partial u} + \frac{d(u + \Delta u, v) - d(u, v)}{\Delta u} n + d \frac{\partial n}{\partial u}
\]
By the definition of partial derivative

\[
\frac{\partial d(u, v)}{\partial u} = \lim_{\Delta u \to 0} \frac{d(u + \Delta u, v) - d(u, v)}{\Delta u}
\]

For small \( \Delta u \), we have that

\[
\frac{\partial p'}{\partial u} = \frac{\partial p}{\partial u} + \frac{d(u + \Delta u, v) - d(u, v)}{\Delta u} n + d \frac{\partial n}{\partial u}
\]

\[
\frac{\partial p'}{\partial v} = \frac{\partial p}{\partial v} + \frac{d(u, v + \Delta v) - d(u, v)}{\Delta v} n + d \frac{\partial n}{\partial v}
\]
- Effect
Bump Mapping

- Effect

Bump Map

without

with
- **Effect**
Bump Mapping

- **Advantages**
  - Nice depth effects
  - Easy to implement
  - Reasonably fast performance

- **Disadvantages**
  - No real p’ is created
  - Does not affect objects surface
    - Does not affect shadow casting process
    - Does not affect objects edges visualization
Bump Mapping

- Effect
Bump Mapping

- Effect
Bump Mapping

- Effect
CS 563 Advanced Topics in Computer Graphics Materials

by Paulo Gonçalves de Barros