Advanced Graphics and Animation

3D Illumination 1

Daniel Jones
Department of Computing
24 November 2010
Aims and Objectives

- In today’s lecture:
  - Light, colour and perception
  - Types of light source
  - Vector maths and lighting
  - Case study: implementing a point light in GLSL
Light, colour and perception

Light

- The light we perceive is a mixture of different lights, scattered and reflected against the surroundings with different material properties.

- Light is a form of energy, and can be described by its wavelength. Colours are the wavelengths of light that are visible to the eye.

- Simply, the colour that a material reflects is perceived as that material’s colour.
Visible Spectrum

The portion of the electromagnetic spectrum that can be detected by the human eye (~380-750nm)
If an object absorbs all wavelengths of light except red, it appears red in a white light.

**Surface Reflectance of a Red Object**

![Graph showing the energy distribution of a red object's reflectance over wavelength. The graph peaks at around 650 nm, indicating a strong reflection in the red spectrum.](graph.png)
Light, colour and perception

Lighting and Shading

- Without shading, we cannot make out the depth or form of a 3D object.

- In the real world, light-material interactions cause each point of a 3D object to have a different shade, giving us vital depth cues.

- We need to consider:
  - Light sources
  - Material properties
  - Location of viewer
  - Surface orientation
Light, colour and perception

Terminology

- When applied to 3D graphics, adding this information is collectively referred to as **shading**.

- Slight distinction:
  - **Illumination**: calculates intensity at a particular point on a surface.
  - **Shading**: uses these calculated intensities to shade the whole surface or the whole scene.
Light, colour and perception

Scene Illumination

- Depends on the light source, the path of the light, how that light interacts with objects, what the properties of those objects are, and the viewpoint for the scene.
Light, colour and perception

Terminology

- When applied to 3D graphics, adding this information is collectively referred to as shading.

- Slight distinction:
  - **Illumination**: calculates intensity at a particular point on a surface.
  - **Shading**: uses these calculated intensities to shade the whole surface or the whole scene.
Light, colour and perception

Determining pixel colour

- The simplest way to think of things is that we need to set each pixel in a scene to a certain colour.

- To determine that colour we need to combine the effects of the lights with the surface properties of the polygon visible in that pixel.
Light, colour and perception

Difficulties

- The process is conceptually easy (though computationally expensive) and we can be quite precise.

- The more precise we are, the more realistic our images will look.

- The physics behind the interaction between lights and material is much more complex. Simulating exact physics is expensive, even if we simplify it.
Light sources

Types of Light Source

- Light can be sent into a scene in a number of ways:
  - a **point source**: illuminates in all directions and diminishes with distance
  - a **directional source**: shines in a particular direction but doesn’t emanate from any particular location
  - a **spotlight**: limited to a small cone-shaped region of the scene
  - **ambient light**: a baseline constant which is everywhere in the scene
1. Ambient

Models a baseline level of light, as if scattered indirectly from the surrounds. A hack, rather than a true light source.
2. Directional

All rays have a common direction but no point of origin, as if the source was infinitely distance.

*eg. the sun*
Light sources

3. Point Source

Emits rays in radial directions from its source.

*eg. light bulb.*
4. Spotlight

A point source whose intensity falls off away from a given direction. Assumed to have a graduated edge: illumination is at its max in the centre.

*eg.* torch
## Light Sources

### Light Source Properties

<table>
<thead>
<tr>
<th>Type of Light</th>
<th>Colour (light/material colour blending)</th>
<th>Direction (specular/diffuse components)</th>
<th>Location (distance attenuation)</th>
<th>Beam (in/out of spot focus)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ambient</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>directional</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>point</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>spot</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>
Vector mathematics and lighting

Vector maths

- Key to implementing and understanding lighting
- Key concepts:
  - vector addition and magnitude
  - unit vectors and normalization
  - dot product and cross product
  - the surface normal
Vector maths refresher

- \((A_1, A_2) + (B_1, B_2) = (A_1 + B_1, A_2 + B_2)\)

A \((3, 1)\)
B \((1, 2)\)
C \((4, 3)\)
Vector mathematics and lighting

Vector maths refresher

- $|A| = \sqrt{A_1 \times A_1 + A_2 \times A_2}$
- $= \text{length of vector } A$

$C = (4, 3)$

$|C| = \sqrt{16 + 9}$

$= \sqrt{25}$

$= 5$
Vector mathematics and lighting

Vector maths refresher

- Unit vector always has length (magnitude) = 1
- ...so divide a vector by its magnitude: normalization

\[ \hat{C} = \frac{C}{|C|} \]

\[ C = (4, 3) \]
\[ |C| = 5 \]
\[ \hat{C} = (0.8, 0.6) \]
Vector maths refresher

- **dot product**: the angle between two vectors multiplied by their magnitudes

\[ \mathbf{A} \cdot \mathbf{B} = |\mathbf{A}| |\mathbf{B}| \cos(\theta) \]

*if \(|\mathbf{A}| = |\mathbf{B}| = 1\):* \[ \mathbf{A} \cdot \mathbf{B} = \cos(\theta) \]
Vector maths refresher

- **cross product**: gives a vector perpendicular to both A and B

\[ \mathbf{C} = \mathbf{A} \times \mathbf{B} \]

- A (3, 1)
- B (1, 2)
Normals

- A normal is a vector **orthogonal** to a plane
- Given by the vector **cross product** of any two edges of the plane or polygon
- In 3D graphics, often used to determine orientation towards a light source.
## Case study: Point light in GLSL

### Point light in GLSL

<table>
<thead>
<tr>
<th>Type of light</th>
<th>Colour (light/material colour blending)</th>
<th>Direction (specular/diffuse components)</th>
<th>Location (distance attenuation)</th>
<th>Beam (in/out of spot focus)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ambient</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>directional</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>point</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>spot</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>
Case study: Point light in GLSL

Vectors involved in lighting

Position

Light

Normal

Reflection

View
Case study: Point light in GLSL

How to implement lighting

- **Vertex shader:**
  - calculates per-vertex normals, lighting and reflectance angle, viewing angle
  - thus the light intensity for this vertex

- **Fragment shader:**
  - receives interpolated colour and light intensity values based on vertex calculations
  - simply has to apply them to the given pixel!
Two components to our directional light:

- **diffuse**
  general scattered light

  The amount of diffuse light on a surface depends on how close the normal is to the light’s angle of incidence.

- **specular**
  reflected light

  The amount of specular light depends on how close the viewer’s position is to the angle of reflection.
Case study: Point light in GLSL

1. Vertex shader

```glsl
// focal factor for specular highlights (positive floats)
uniform float SpecularFocus;

// scaling factors for the amount of specular and diffused lighting
uniform float SpecularContribution;
uniform float DiffuseContribution;

// variables passed through to fragment shader
varying float lightIntensity;

// position of our fixed light source, in eye co-ordinates
// const vec3 lightPos = vec3(200, 200, 50);
uniform vec3 lightPos;
```
Case study: Point light in GLSL

1. Vertex shader

```cpp
void main()
{
    // calculate the vertex position in eye coordinates
    vec3 vertexEyePos = vec3(gl_ModelViewMatrix * gl_Vertex);

    // surface normal: must be transformed to EC using the NormalMatrix
    vec3 vertexNormal = normalize(gl_NormalMatrix * gl_Normal);

    // unit vector pointing from the vertex to the light source
    vec3 lightDir = normalize(lightPos - vec3(vertexEyePos));

    // calculate the vector corresponding to the light source reflected in the vertex surface.
    // lightDir is negated as GLSL expects an incoming (rather than outgoing) vector
    vec3 lightReflection = reflect(-lightDir, vertexNormal);

    // unit vector pointing from the vertex towards the viewer position
    vec3 vertexViewDir = normalize(-vertexEyePos);
}```
Case study: Point light in GLSL

1. Vertex shader

```glsl
// lambert: diffused light is dot product of light direction and vertex normal. 
// anything beyond 90 degrees receives no diffused light (hence the max(N, 0.0))
float amountDiffuse = max(dot(lightDir, vertexNormal), 0.0);

// specular light is dot product of light reflection vector and our viewing vector. 
// the closer we are to the reflection angle, the greater the specular sheen.
float amountSpecular = max(dot(lightReflection, vertexViewDir), 0.0);

// apply an additional pow() to focus the specular effect 
// (try playing with this value)
amountSpecular = pow(amountSpecular, SpecularFocus);

// calculate actual light intensity
lightIntensity = SpecularContribution * amountSpecular + 
    DiffuseContribution * amountDiffuse;

// set the vertex's screen position.
gl_Position = ftransform();

// pass through the color specified in GL vertex attributes
gl_FrontColor = gl_Color;
```
Case study: Point light in GLSL

2. Fragment shader

```glsl
// fragment shader: apply light intensity to pixels.

// variables passed through from vertex shader
varying float lightIntensity;

void main()
{
    vec3 color = vec3(gl_Color) * lightIntensity;
    gl_FragColor = vec4(color, 1.0);
}
```
Case study: Point light in GLSL

The finished product
Conclusion

We have covered:

‣ This time:

‣ General lighting and perception issues
‣ Types of light source
‣ Vector algebra for lighting
‣ The fundamentals of implementing lights in GLSL
Conclusion

Still to cover:

- Next time:
  - OpenGL fixed lighting functionality
  - Processing vs OpenGL vs GLSL lighting
  - Radiosity and ray-tracing
Further Resources

- **The Orange Book ("OpenGL Shading Language", Rost, 2006)**
  Chapter 6: Simple Shading Example
  Chapter 9: Emulating OpenGL Fixed Functionality
  Chapter 12: Lighting


- **Greg Sidelnikov’s comprehensive OpenGL lighting tutorial**