Color, Lighting

305890
Spring 2011
4/18/2011
Kyoung Shin Park

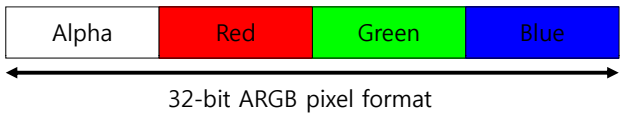
Overview

- Representing Color
- Vertex Colors
- Flat Shading, Gouraud Shading, Phong Shading
- Lighting
  - Light-Material Interaction
  - Diffuse/Ambient/Specular Lighting
  - Light sources: Point Light/Directional Light/Spot Light

Representing Colors

- Color structure
  - 4 (Red, Green, Blue, Alpha) value

```
public struct Color : IPackedVector<uint>,IPackedVector,IEquatable<Color>
{
    public Color(Vector3 vector);
    public Color(Vector4 vector);
    public Color(float r, float g, float b);
    public Color(float r, float g, float b, float a);
    public static Color AliceBlue { get; }
    public static Color Cyan { get; }
....
```



32-bit ARGB pixel format

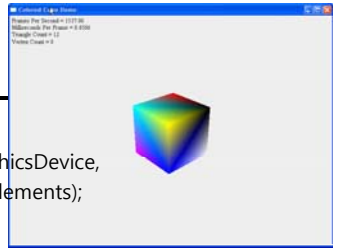
VertexPositionColor

- Vertex structure include position and color to each vertex

```
public struct VertexPositionColor
{
    public Color Color;
    public Vector3 Position;
    public static readonly VertexElement[] VertexElements;

    public VertexPositionColor(Vector3 position, Color color);
...
};
```

Color Cube Demo



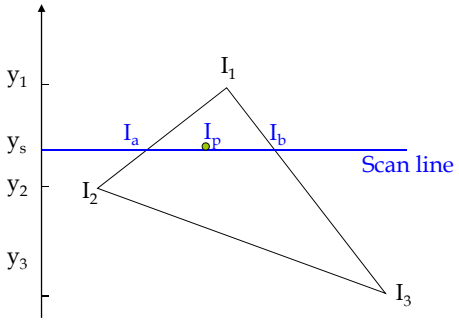
```
// vertex declaration
vertexDeclaration = new VertexDeclaration(graphics.GraphicsDevice,
                                          VertexPositionColor.VertexElements);
// create vertex
vertices = new VertexPositionColor[8];
vertices[0] = new VertexPositionColor(new Vector3(-1.0f, -1.0f, 1.0f), Color.White);
vertices[1] = new VertexPositionColor(new Vector3(-1.0f, 1.0f, 1.0f), Color.Black);
vertices[2] = new VertexPositionColor(new Vector3(1.0f, 1.0f, 1.0f), Color.Red);
vertices[3] = new VertexPositionColor(new Vector3(1.0f, -1.0f, 1.0f), Color.Green);
vertices[4] = new VertexPositionColor(new Vector3(-1.0f, -1.0f, -1.0f), Color.Blue);
vertices[5] = new VertexPositionColor(new Vector3(-1.0f, 1.0f, -1.0f), Color.Yellow);
vertices[6] = new VertexPositionColor(new Vector3(1.0f, 1.0f, -1.0f), Color.Cyan);
vertices[7] = new VertexPositionColor(new Vector3(1.0f, -1.0f, -1.0f), Color.Magenta);
// initialize VB allocating memory for each vertex
vertexBuffer = new VertexBuffer(graphics.GraphicsDevice,
    VertexPositionColor.SizeInBytes * (vertices.Length), BufferUsage.None);
// set VB data to the array of vertices
vertexBuffer.SetData<VertexPositionColor>(vertices);
```

Shading

- Shading is the process of determining the colors of the pixels in a primitive.
  - Flat shading: fills the polygon face with the first vertex color
  - Gouraud shading (smooth shading): interpolates the polygon face with the colors at each vertex (Default)

Gouraud Shading

- Gouraud shading
  - The color of a pixel is linearly interpolated using the colors of all vertices in the primitives



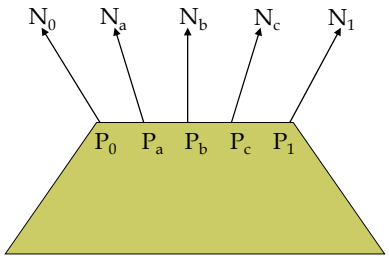
$$I_a = I_1 - (I_1 - I_2)\frac{y_1 - y_s}{y_1 - y_2}$$

$$I_b = I_1 - (I_1 - I_3)\frac{y_1 - y_s}{y_1 - y_3}$$

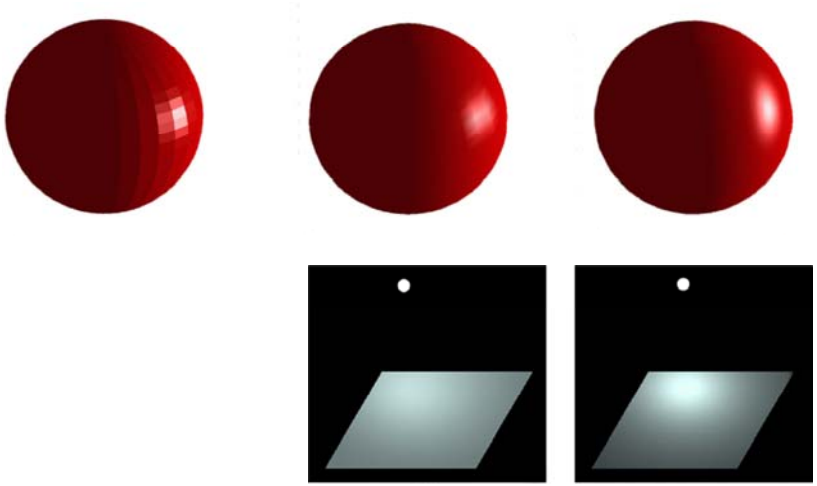
$$I_p = I_b - (I_b - I_a)\frac{x_b - x_p}{x_b - x_a}$$

Phong Shading

- Normal-vector interpolated shading
- Phong shading computes the normal vector for each pixel on the polygon



Flat, Gouraud, and Phong Shading

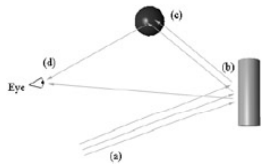


Lighting

- Light Components
- Materials
- Vertex Normals
- Light Sources

Light and Material Interaction

- When using lighting, we no longer specify vertex colors ourselves; rather, we specify materials and lights, and then, apply a lighting equation, which computes the vertex colors for us based on light/material interaction.
- Materials can be thought of as the properties that determine how light interacts with an object.
- We model lights by an additive mixture of red, green, and blue light (RGB); we can simulate many light colors.



Lighting Component

- Light
  - Ambient light – Ambient light comes from no particular direction. It reflects equally in all directions.
  - Diffuse light – The basic lighting effects is diffuse lighting. The intensity of diffuse lighting depends on the orientation of the surface relative to the light source. Diffuse light is reflected equally in all directions.
  - Specular light – Specular light is the light that is directly reflected off the surface to the camera. Its intensity depends on the orientation of the surface relative to camera, as well as to the light source.

Lighting Component

- Light source color representation using Color structure

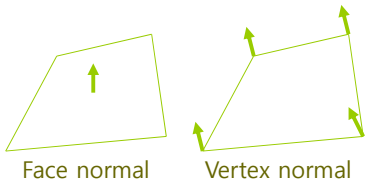
```
Color redAmbient(1.0f, 0.0f, 0.0f, 1.0f);    // alpha is not used in light
Color blueDiffuse(0.0f, 0.0f, 1.0f, 1.0f);
Color whiteSpecular(1.0f, 1.0f, 1.0f, 1.0f);
```

Materials

- Material properties define how a surface reflects light. Basically, they represent the surface color.
- *When lighting is active, the material is used instead of color.*
- MATERIAL
  - Surface diffuse/ambient/specular reflections
  - Emissive material (to make object appeared to be self-luminous)
  - Sharpness of specular reflection (higher value, smaller highlights)

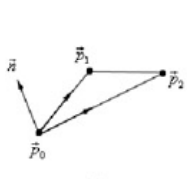
Vertex Normal

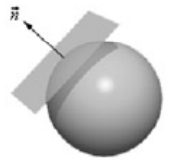
- Normal
  - Lighting computation uses vertex normals
- Vertex structure
  - Use normals instead of colors

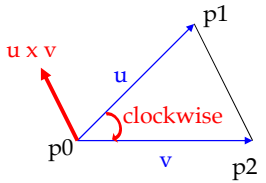


Vertex Normal

- Face Normal
  - Compute the normal vector of a triangle consisting of vertex p0, p1, p2



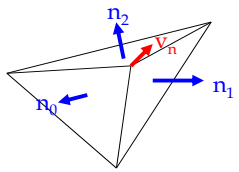


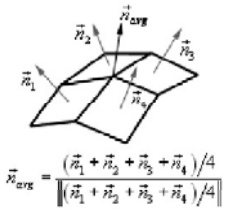


Vertex Normal

- Vertex Normal
  - Vertex normal calculation using adjacent faces

$$v_n = \frac{1}{3}(n_0 + n_1 + n_2)$$

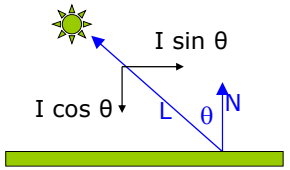




$$\vec{n}_{avg} = \frac{(\vec{n}_1 + \vec{n}_2 + \vec{n}_3 + \vec{n}_4)/4}{\|(\vec{n}_1 + \vec{n}_2 + \vec{n}_3 + \vec{n}_4)/4\|}$$

Diffuse Lighting

- Lambert's Cosine Law
  - θ between the normal and light vector
  - Maximum intensity when the normal and light vector are perfectly aligned (θ = 0)
  - f(θ) = max(cosθ, 0) = max(L • N, 0)

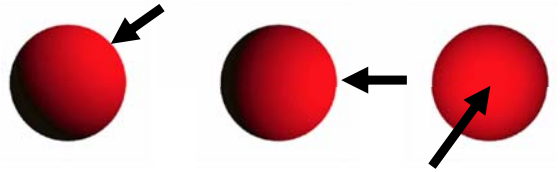


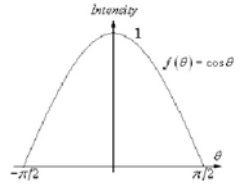
$$Diffuse\ Reflection \propto \cos\theta$$

$$Diffuse\ Reflection = K_d I_d \cos\theta = K_d I_d (N \bullet L)$$

Light vector
Surface normal vector

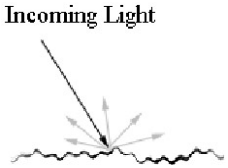
I_d: diffuse light intensity
K_d: diffuse light coefficient





Diffuse Lighting

- Diffuse lighting calculation (viewpoint independent)



$$\max(L \bullet N, 0) \bullet (L_d \otimes M_d)$$

L_d: diffuse light color
M_d: diffuse material color
L: light vector
N: vertex normal vector

Ambient Lighting

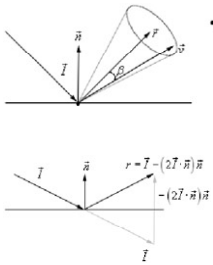
- Ambient lighting calculation

$$(L_a \otimes M_a)$$

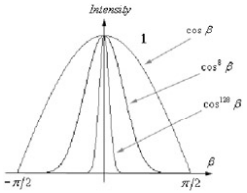
L_a: ambient light color
M_a: ambient material color

Specular Lighting

- Specular lighting calculation (viewpoint dependent)



$$\cdot \vec{E} \quad (\max(V \bullet R, 0))^p \bullet (L_s \otimes M_s)$$



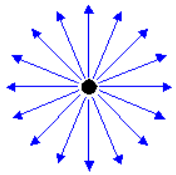
L_s: specular light color
M_s: specular material color
V: view vector
R: light reflection vector, R = L – (2L • N)N

Point/Directional/Spot Light

- Light sources
  - Point light
  - Directional light
  - Spot light

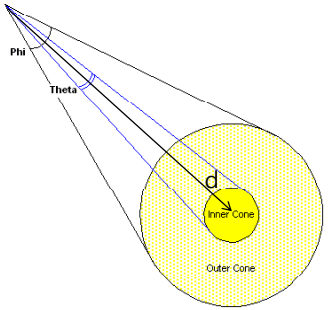
$$L = \frac{(S-P)}{\|S-P\|}$$

L: light vector
S: point/directional light position
P: vertex position



Point light

Directional light



Spot light

Light Attenuation

- Light attenuation
  - Light intensity weakness as a function of distance based on the inverse square law.

$$I(d) = \frac{I_0}{d^2}$$

$$I(d) = \frac{I_0}{a_0 + a_1 d + a_2 d^2}$$

$$d = \|S - P\| (i.e., the\ distance\ between\ P\ and\ S)$$

Lighting with the BasicEffect class

- Default Lighting
  - effect.EnableDefaultLighting();
- Custom Lighting
  - effect.LightEnabled = true;
  - effect.DirectionalLight0.DiffuseColor = new Vector3(1, 0, 0); // red
  - effect.DirectionalLight0.Direction = new Vector3(1, 0, 0);
  - effect.DirectionalLight0.SpecularColor = new Vector3(1, 1, 0); // yellow
  - effect.DirectionalLight0.Enabled = true; // can turn individual lights on/off
  - effect.AmbientLightColor = new Vector3(0.2f, 0.2f, 0.2f);
  - effect.EmissiveColor = new Vector3(1, 0, 0);
  - effect.PreferPerPixelLighting = true; // per-pixel lighting

HLSL: Ambient Light

```
struct VertexShaderInput {
    float4 Position : POSITION0;
};
struct VertexShaderOutput {
    float4 Position : POSITION0; // Transform to homogeneous clip space.
};
VertexShaderOutput VertexShaderFunction(VertexShaderInput input)
{
    VertexShaderOutput output;
    float4 worldPosition = mul(input.Position, World);
    float4 viewPosition = mul(worldPosition, View);
    output.Position = mul(viewPosition, Projection);
    return output;
}
float4 PixelShaderFunction(VertexShaderOutput input) : COLOR0
{
    float ambientIntensity = 0.8f;
    float4 ambientColor = float4(1.0f, 0.0f, 0.0f, 1.0f);
    return ambientIntensity * ambientColor;
}
```

HLSL: Ambient and Diffuse Light

```
struct VertexShaderInput
{
    float4 Position : POSITION0;
    float3 Normal : NORMAL0;
};
struct VertexShaderOutput
{
    float4 Position : POSITION0;        // transformed position
    float3 Normal : TEXCOORD1;          // transformed & normalized surface normal vector
};
VertexShaderOutput VertexShaderFunction(VertexShaderInput input)
{
    VertexShaderOutput output;
    float4 worldPosition = mul(input.Position, World);
    float4 viewPosition = mul(worldPosition, View);
    output.Position = mul(viewPosition, Projection);
    output.Normal = normalize(mul(input.Normal, World));
    return output;
}
```

HLSL: Ambient and Diffuse Light

```
float4 PixelShaderFunction(VertexShaderOutput input) : COLOR0
{
    float ambientIntensity = 0.8f;
    float4 ambientColor = float4(0.2f, 0.2f, 0.2f, 1.0f);
    float4 ambient = ambientIntensity * ambientColor;

    float duffuseIntensity = 1.0f;
    float4 diffuseColor = float4(1.0f, 0.0f, 0.0f, 1.0f);
    float4 diffuse = diffuseIntensity * diffuseColor * saturate(dot(Light, input.Normal));

    return (ambient + diffuse);
}
```

HLSL: Ambient, Diffuse, Specular Light

```
struct VertexShaderInput  {
    float4 Position : POSITION0;
    float3 Normal : NORMAL0;
};
struct VertexShaderOutput {
    float4 Position : POSITION0;        // transformed position
    float3 L : TEXCOORD0;               // light
    float3 N : TEXCOORD1;               // surface normal
    float3 V : TEXCOORD2;               // view
};
VertexShaderOutput VertexShaderFunction(VertexShaderInput input)
{
    VertexShaderOutput output = (VertexShaderOutput) 0;
    float4 worldPos = mul(input.Position, World);
    float4 viewPos = mul(worldPos, View);
    output.Position = mul(viewPos, Projection); // transformed vertex position
    output.N = mul(input.Normal, World); // transformed surface normal
    output.L = LightPosition; // Light direction
    output.V = EyePosition - worldPos; // View direction (Eye position - vertex position)
    return output;
}
```

HLSL: Ambient, Diffuse, Specular Light

```
float4 PixelShaderFunction(VertexShaderOutput input) : COLOR
{
    // normalize our vectors.
    float3 Normal = normalize(input.N);
    float3 LightDir = normalize(input.L);
    float3 ViewDir = normalize(input.V);
    // calculate diffuse light
    float Diffuse = saturate(dot(Normal, LightDir));
    // create our reflection shader
    // R = 2 * (N.L) * N - L
    float3 Reflect = normalize(2 * Diffuse * Normal - LightDir);
    // calculate our specular light
    float Specular = pow(saturate(dot(Reflect, ViewDir)), 20); // R.V^n
    // return our final light equation
    // I = Acolor * Aintensity + Dcolor * Dintensity * N.L + Scolor * Sintensity * (R.V)n
    return AmbientColor + DiffuseColor * Diffuse + SpecularColor * Specular;
}
```