Rendering Thin Film Interference on Soap Bubbles

Mucong Ding

Hong Kong University of Science and Technology

mcding@connect.ust.hk

November 26, 2017

Overview

Section 1

Background Theory

Introduction (I)

What should be a soap bubble look like?

Figure: A Real World Soap Bubble

Introduction (II)

Figure: A Real World Soap Bubbles

The dominant effects

- Reflection
- Refraction with high Fresnel Effect
- Thin film interference (colors)

The third effect is usually omitted by CG simulation.

Introduction (III)

How to simulate thin film interference?

- Go deeper in physics
- Design workable shader algorithm

Fresnel Equation

The well-know Fresnel shading equation in CG is designed based on the Fresnel Equation.

- $R_{\mathrm{S}} = \frac{n_1 \cos\theta - n_2\sqrt{1-(\frac{n_1}{n_2}\sin\theta)^2}}{n_1 \cos\theta + n_2\sqrt{1-(\frac{n_1}{n_2}\sin\theta)^2}}$
- $R_{\mathrm{P}} = \frac{n_1 \sqrt{1-(\frac{n_1}{n_2}\sin\theta)^2} - n_2\cos\theta}{n_1 \sqrt{1-(\frac{n_1}{n_2}\sin\theta)^2} + n_2\cos\theta}$
- Reflectance as function of refractive indices ratio n_1/n_2 and incident angle θ
- Polarization matters, S and P polarization components of lights

Thin Film Effect

Thin film: two interfaces with thickness $d \sim 1000$ nm.

- Refracted light cancel or reinforce depends on the thickness-to-wavelength ratio d/λ
- Approximate effective reflectance derived by applying Fresnel equation twice
- $R(\lambda, \theta, d) = 2R_{\mathrm{P}}^2 \frac{1-\cos\delta}{1+R_{\mathrm{P}}^4-2R_{\mathrm{P}}^2\cos\delta} + 2R_{\mathrm{S}}^2 \frac{1-\cos\delta}{1+R_{\mathrm{S}}^4-2R_{\mathrm{S}}^2\cos\delta}$
- $\delta(\lambda, \theta, d) = 4\pi n \frac{d}{\lambda} \cos\theta$
- Ignore polarization (by averaging) since when don't have enough information

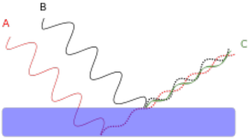


Figure: Thin Film Interference

The Shading Equation

Idea: Effective reflection and transmission as if only one interface:

- $L_P(\lambda) = (1 - R(\lambda, \theta, d))L_{it}(\lambda) + R(\lambda, \theta, d)L_{ir}(\lambda)$
- Reflectance $R(\lambda, \theta, d)$ defined on the previous slide
- Wave-length λ matters!

Section 2

Live Demo

Demonstration

Demo first!

- http://mcding.student.ust.hk/comp5411/

Summary of Observations

What we have:

- Colors! (thin film interference)
- Fresnel effect originated from Fresnel equation itself!

What we don't have:

- Only texture mapping, no good light sources (may magnify interference and Fresnel effects)
- Good simulation of transmission through the bubble (4 interfaces) (approximated by an effective refractive index $n_e \sim 1.005$ now)

Section 3

The Shaders

Color and Wavelength

The shading equation depends on wave-length λ, how can we obtain it?

- Assigning approximate wave-lengths to each of the RGB components
- R $\sim$ 660 nm, G $\sim$ 510 nm, B $\sim$ 450 nm
- Treat them differently from start to end
- Code:

```
        uniforms: {
            mWaveLength: { type:"fv1", value: [660, 510, 450] },
```

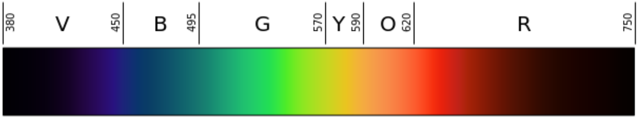
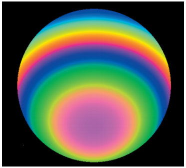


Figure: Spectral Colors

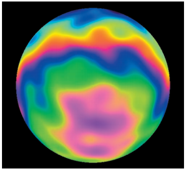
Thickness Distribution

Uniform thickness d on the surface is not physical, makes interference colors only appears when incident angles is close to 90 deg.
Simulate the thickness distribution on the globe! Two main effects

- Drifting: by gravity
- Sloshing: by perturbation



(a) Only Drifting



(b) Add Sloshing

Code:

```
"float direction = fract(sin(dot(normalize(worldNormal).xy, vec2(1.0, 1.0)) * PI));",
"float thickness = mAverageThickness * (1.0 - normalize(worldNormal).z * direction  * mThicknessRange);",
```

Following the Physics (I)

The major part of shaders following the physics equations.

Vertex Shader:

```
        "void main() {",

        "vec4 mvPosition = modelViewMatrix * vec4( position, 1.0 );",
        "vec4 worldPosition = modelMatrix * vec4( position, 1.0 );",
        "vec3 worldNormal = normalize(mat3(modelMatrix[0].xyz, modelMatrix[1].xyz, modelMatrix[2].xyz) * normal);",
        "vec3 I = worldPosition.xyz - cameraPosition;",

        "vReflect = reflect(I, worldNormal);",
        "vRefract = refract(normalize(I), worldNormal, mEffectiveRelactiveIndex);",

        "float cosine = dot(normalize(I), normalize(worldNormal));",
        "float sine = sqrt(1.0 - cosine * cosine);",
        "float effectiveCosine = sqrt(1.0 - sine * sine / mRefractiveIndex / mRefractiveIndex);",
        "float reflectivityS = (cosine - mRefractiveIndex * effectiveCosine)/(cosine + mRefractiveIndex * effectiveCosine);",
        "reflectivityS = reflectivityS * reflectivityS;",
        "float reflectivityP = (effectiveCosine - mRefractiveIndex * cosine)/(effectiveCosine + mRefractiveIndex * cosine);",
        "reflectivityP = reflectivityP * reflectivityP;",

        "float direction = fract(sin(dot(normalize(worldNormal).xy, vec2(1.0, 1.0)) * PI));",
        "float thickness = mAverageThickness * (1.0 - normalize(worldNormal).z * direction  * mThicknessRange);",

        "float cosineDelta[3];",
        "cosineDelta[0] = cos(4.0 * PI * mRefractiveIndex * thickness / mWaveLength[0] * cosine);",
        "cosineDelta[1] = cos(4.0 * PI * mRefractiveIndex * thickness / mWaveLength[1] * cosine);",
        "cosineDelta[2] = cos(4.0 * PI * mRefractiveIndex * thickness / mWaveLength[2] * cosine);",

        "reflectivity[0] = 2.0 * reflectivityS * reflectivityS * (1.0 - cosineDelta[0]) / " +
        "(1.0 + reflectivityS * reflectivityS * reflectivityS * reflectivityS - 2.0 * reflectivityS * reflectivityS * cosineDelta[0]);",
        "reflectivity[0] += 2.0 * reflectivityP * reflectivityP * (1.0 - cosineDelta[0]) / " +
        "(1.0 + reflectivityP * reflectivityP * reflectivityP * reflectivityP - 2.0 * reflectivityP * reflectivityP * cosineDelta[0]);",
        "reflectivity[1] = 2.0 * reflectivityS * reflectivityS * (1.0 - cosineDelta[1]) / " +
        "(1.0 + reflectivityS * reflectivityS * reflectivityS * reflectivityS - 2.0 * reflectivityS * reflectivityS * cosineDelta[1]);",
        "reflectivity[1] += 2.0 * reflectivityP * reflectivityP * (1.0 - cosineDelta[1]) / " +
        "(1.0 + reflectivityP * reflectivityP * reflectivityP * reflectivityP - 2.0 * reflectivityP * reflectivityP * cosineDelta[1]);",
        "reflectivity[2] = 2.0 * reflectivityS * reflectivityS * (1.0 - cosineDelta[2]) / " +
        "(1.0 + reflectivityS * reflectivityS * reflectivityS * reflectivityS - 2.0 * reflectivityS * reflectivityS * cosineDelta[2]);",
        "reflectivity[2] += 2.0 * reflectivityP * reflectivityP * (1.0 - cosineDelta[2]) / " +
        "(1.0 + reflectivityP * reflectivityP * reflectivityP * reflectivityP - 2.0 * reflectivityP * reflectivityP * cosineDelta[2]);",
        "reflectivity[0] *= mInterferenceMagnifier;",
        "reflectivity[1] *= mInterferenceMagnifier;",
        "reflectivity[2] *= mInterferenceMagnifier;",

        "gl_Position = projectionMatrix * mvPosition;",

        "}"
```

Following the Physics (II)

Fragment Shader:

```
"void main() {",

"vec4 reflectedColor = vec4(1.0);",
"reflectedColor.r = textureCube(tCube, vec3(-vReflect.x, vReflect.yz)).r;",
"reflectedColor.g = textureCube(tCube, vec3(-vReflect.x, vReflect.yz)).g;",
"reflectedColor.b = textureCube(tCube, vec3(-vReflect.x, vReflect.yz)).b;",
"vec4 refractedColor = vec4(1.0);",
"refractedColor.r = textureCube(tCube, vec3(-vRefract.x, vRefract.yz)).r;",
"refractedColor.g = textureCube(tCube, vec3(-vRefract.x, vRefract.yz)).g;",
"refractedColor.b = textureCube(tCube, vec3(-vRefract.x, vRefract.yz)).b;",
"vec4 color = vec4(1.0);",
"color.r = mix(refractedColor.r, reflectedColor.r, clamp(reflectivity[0], .0, 1.0));",
"color.g = mix(refractedColor.g, reflectedColor.g, clamp(reflectivity[1], .0, 1.0));",
"color.b = mix(refractedColor.b, reflectedColor.b, clamp(reflectivity[2], .0, 1.0));",

"gl_FragColor = color;",

"}"
```

Section 4

Challenges, Result, and Future Work

Challenges and Result

Challenges:

- Deriving approximated shading equation from real physics (partially aided by reference [2])
- Assigning RGB components with different wave-lengths to simulate compound color light
- Simulating drifting and sloshing which affect thickness distribution
- Dynamic texture mapping and setting up the scene (with the help of skeleton example [1])

Result: All challenges solved, as summarized before.

Future Work and Reference

Future Work:

- Adding light source
- Ray tracing
- Distortion matters

Reference:

REFERENCES

[1] three.js, "three.js bubble demo with fresnel effect," "https://threejs.org/examples/webgl_materials_shaders_fresnel.html", 10 2017, [Online; accessed 09-Nov-2017].

[2] K. Iwasaki, K. Matsuzawa, and T. Nishita, "Real-time rendering of soap bubbles taking into account light interference," in *Proceedings Computer Graphics International, 2004.*, June 2004, pp. 344–348.

[3] A. Glassner, "Soap bubbles. 2 [computer graphics]," *IEEE Computer Graphics and Applications*, vol. 20, no. 6, pp. 99–109, Nov 2000.