

# **Information Visualization**

W08: Exercise - Shader Programming

Graduation School of System Informatics

Department of Computational Science

**Naohisa Sakamoto, Akira Kageyama**

May.10, 2017

# Schedule

- W01 4/11 Guidance
- W02 4/12 Setup
- W03 4/18 Introduction to Data Visualization
- W04 4/19 CG Programming
- W05 4/25 Rendering Pipeline
- W06 4/26 Coordinate Systems and Transformations
- W07 5/09 Shading
- W08 5/10 Shader Programming
- W09 5/16 Visualization Pipeline
- W10 5/17 Data Model and Transfer Function
- W11 5/23 Scalar Data Visualization 1 (Isosurface Extraction)
- W12 5/24 Implementation of Isosurface Extraction
- W13 5/30 Scalar Data Visualization 2 (Volume Rendering)
- W14 5/31 Implementation of Volume Rendering
- W15 6/06 Student Presentations

# GLSL

- Types
  - void, bool, int, float
  - vec2, vec3, vec4
  - bvec2, bvec3, bvec4
  - ivec2, ivec3, ivec4
  - mat2, mat3, mat4
  - sampler2D, sampler3D
  - ...

# GLSL

- Vector Components
  - $\{x, y, z, w\}$  represent points or normals  
e.g.: `pos.x`, `pos.y`, `pos.z`, `pos.w`
  - $\{r, g, b, a\}$  represent colors  
e.g.: `col.r`, `col.g`, `col.b`, `col.a`
  - $\{s, t, p, q\}$  represent texture coordinates  
e.g.: `tex.s`, `tex.t`, `tex.p`, `tex.q`
  - Swizzled and replicated  
e.g.: `pos.xy`, `pos.xx`, `pos.zy`, ...

# GLSL

- Matrix Components

- Column-major

- Constructors

- e.g.: `mat2(float)`

- `mat2(vec2, vec2)`

- `mat2(float, float, float, float)`

- Access components of a matrix with array subscripting syntax

- e.g.) `mat4 m;`

- `m[1] = vec4(2.0);`

- `m[0][0] = 1.0;`

# GLSL

- Qualifiers
  - `const` constant parameter
  - `attribute` input value of vertex shader
  - `uniform` global value
  - `varying` output value of vertex shader  
input value of fragment shader

# GLSL

- Built-In Inputs and Outputs (Vertex Shader)
  - Input
    - `gl_Position` transformed vertex position
    - `gl_PointSize` transformed point size

# GLSL

- Built-In Inputs and Outputs (Fragment Shader)

- Inputs

gl_FragCoord	fragment position (framebuffer)
--------------	------------------------------------

`gl_FrontFacing` fragment belongs to a front-facing primitive

gl\_PointCoord    fragment position (point)

- Outputs

gl\_FragColor      fragment color

```
gl_FragData[n]  fragment color for color
                 attachment n
```



# GLSL

- Built-In Functions
  - Angle and Trigonometry Functions
    - `T radians(T degrees)`
    - `T degrees(T radians)`
    - `T sin(T angle)`
    - `T cos(T angle)`
    - `T tan(T angle)`
    - ...

# GLSL

- Built-In Functions
  - Exponential Functions
    - `T pow(T x, T y)`
    - `T exp(T x)`
    - `T log(T x)`
    - `T sqrt(T x)`
    - ...

# GLSL

- Built-In Functions

- Common Functions

- `T abs(T x)`
    - `T floor(T x), T ceil(T x)`
    - `T min(T x, T y), T max(T x, T y)`
    - `T clamp(T x, T minVal, T maxVal)`
    - `T mix(T x, T y, T a)`
    - ...

# GLSL

- Built-In Functions
  - Geometric Functions
    - `T length(T x)`
    - `float distance(T p0, T p1)`
    - `float dot(T x, T y)`
    - `vec3 cross(vec3 x, vec3 y)`
    - `T normalize(T x)`
    - `T reflect(T l, T n)`
    - ...

# Three.js

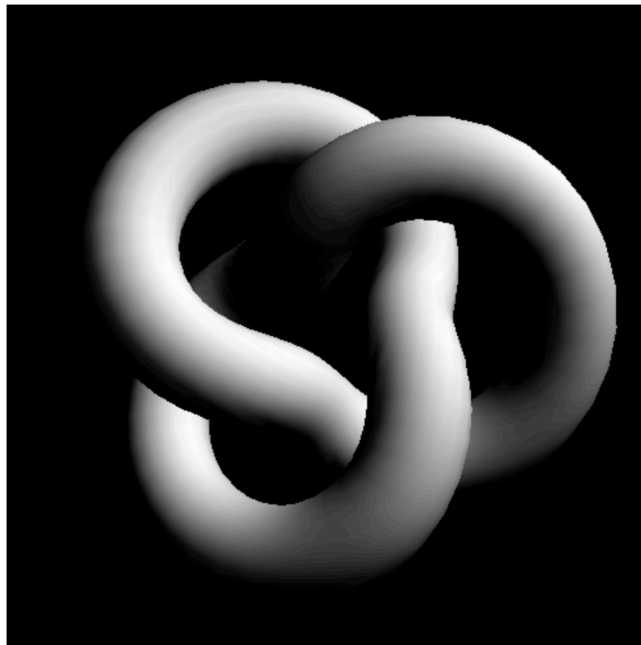
- Built-in uniforms and attributes
  - Vertex Shader
    - `uniform mat4 modelMatrix;`
    - `uniform mat4 modelViewMatrix;`
    - `uniform mat4 projectionMatrix;`
    - `uniform mat4 viewMatrix;`
    - `uniform mat3 normalMatrix;`
    - `uniform vec3 cameraPosition;`
    - `attribute vec3 position;`
    - `attribute vec3 normal;`
    - ...

# Three.js

- Built-in uniforms and attributes
  - Fragment Shader
    - `uniform mat4 viewMatrix;`
    - `uniform vec3 cameraPosition;`

# Torus Knot

- Download files named as main01.js and w08\_ex01.html
- Open w08\_ex01.html with your web browser



# Torus Knot with Three.js

- THREE.TorusKnotGeometry

```
var geometry = new THREE.TorusKnotGeometry(1, 0.3, 100,20);  
var material = new THREE.MeshLambertMaterial();  
  
var torus_knot = new THREE.Mesh( geometry, material );  
scene.add( torus_knot );
```



# Torus Knot with Shaders

- Download files named as main02.js and w08\_ex02.html
- Open w08\_ex02.html with your web browser



# Simple Shaders

- Describes shaders with `<script>`
  - Vertex Shader

```
<script type="x-shader/x-vertex" id="shader.vert">
void main()
{
    gl_Position = projectionMatrix * modelViewMatrix *
vec4( position, 1.0 );
}
</script>
```

# Simple Shaders

- Describes shaders with `<script>`
  - Fragment Shader

```
<script type="x-shader/x-fragment" id="shader.frag">
void main(
{
    gl_FragColor = vec4( 1.0, 1.0, 1.0, 1.0 );
}
</script>
```

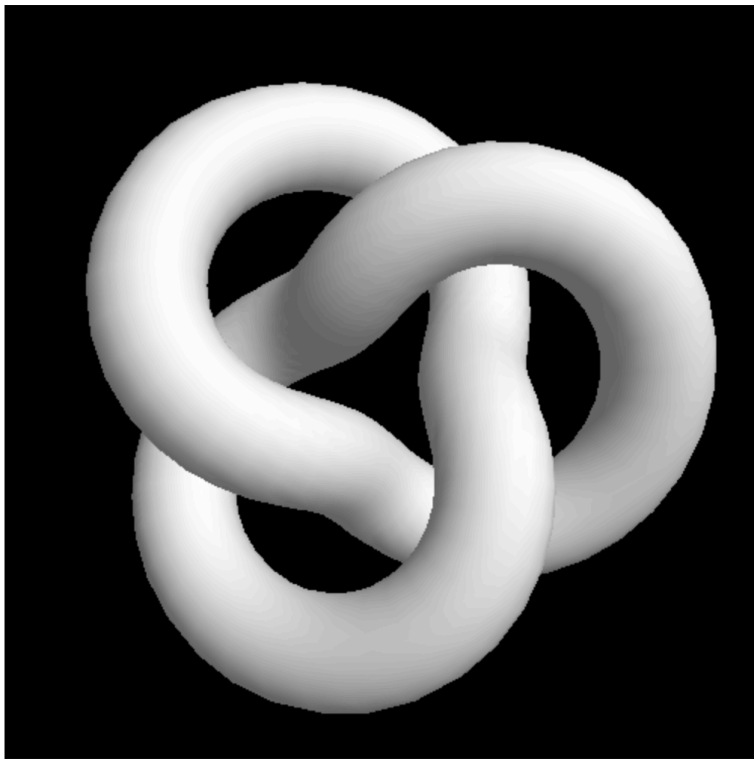
# Simpler Shaders

- Uses THREE.ShaderMaterial instead of THREE.MeshLambertMaterial

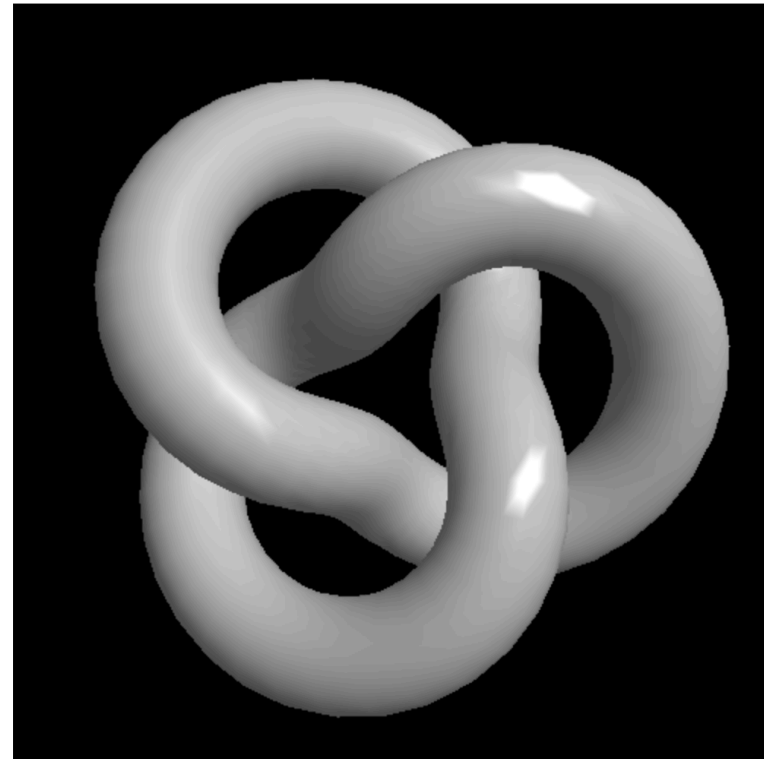
```
var material = new THREE.ShaderMaterial({  
    vertexColors: THREE.VertexColors,  
    vertexShader: document.getElementById('shader.vert').text,  
    fragmentShader: document.getElementById('shader.frag').text  
});
```

# Task 1

- Implement Gouraud shading for the rotating torus knot geometry



Lambertian reflection model



Phong reflection model

# Hint

- Vertex Shader

```
<script type="x-shader/x-vertex" id="gouraud.vert">
    varying vec3 point_color;
    varying vec4 point_position;
    varying vec3 normal_vector;
    uniform vec3 light_position;

    // LambertianReflection function here

    void main()
    {
        point_position = modelViewMatrix * vec4( position, 1.0 );
        normal_vector = normalMatrix * normal;

        vec3 C = color;
        vec3 L = normalize( light_position - point_position.xyz );
        vec3 N = normalize( normal_vector );
        point_color = LambertianReflection( C, L, N );
        gl_Position = projectionMatrix * point_position;
    }
</script>
```

# Hint

- Lambertian Reflection

```
vec3 LambertianReflection( vec3 C, vec3 L, vec3 N )
{
    float ka = 0.4;
    float kd = 0.6;

    float dd = max( dot( N, L ), 0.0 );
    float Ia = ka;
    float Id = kd * dd;
    return C * ( Ia + Id );
}
```

# Hint

- Phong Reflection

```
vec3 PhongReflection( vec3 C, vec3 L, vec3 N )
{
    float ka = 0.3;
    float kd = 0.5;
    float ks = 0.8;
    float n = 50.0;

    vec3 R = reflect( -L, N );
    float dd = max( dot( N, L ), 0.0 );
    float ds = pow( max( dot( R, V ), 0.0 ), n );
    if ( dd <= 0.0 ) { ds = 0.0; }

    float Ia = ka;
    float Id = kd * dd;
    float Is = ks * ds;
    return C * ( Ia + Id + Is );
}
```



# Hint

- Fragment Shader

```
<script type="x-shader/x-fragment" id="gouraud.frag">
    varying vec3 point_color;

    void main()
    {
        gl_FragColor = vec4( point_color, 1.0 );
    }
</script>
```

# Hint

- ShaderMaterial

```
var material = new THREE.ShaderMaterial({  
  vertexColors: THREE.VertexColors,  
  vertexShader: document.getElementById('gouraud.vert').text,  
  fragmentShader: document.getElementById('gouraud.frag').text,  
  uniforms: {  
    light_position: { type: 'v3', value: light.position }  
  }  
});
```

Uniform types:

- i      int      int
- f      float      float
- v2      THREE.Vector2      vec2
- v3      THREE.Vector3      vec3
- v4      THREE.Vector4      vec4
- c      THREE.Color      vec3
- ...

<https://github.com/mrdoob/three.js/wiki/Uniforms-types>

# Task 2

- Implement Phong shading for the rotating torus knot geometry
- Compare the rendering result with Gouraud shading

# Advanced Tasks

- Implement shaders based on the following reflection models
  - Blinn-Phong reflection (Task 3)
  - Cook-Torrance reflection (Task 4)
  - Toon reflection (Task 5)

# Polling

- Take the poll
  - Student ID Number
  - Name
  - URL to Task 1
  - URL to Task 2
  - URL to Task 3 (advanced)
  - URL to Task 4 (advanced)
  - URL to Task 5 (advanced)