MVD: Engine Programming

03 - Light and Camera components

alunthomas.evans@salle.url.edu

laSalle ENG
Universitat Ramon Llull

Parsers class

Biggest change in code-base this week.

It has **static functions** to parse any data

What are static functions/variables in C++?

Static functions/variables

"A static data member is not part of the subobjects of a class. There is only one copy of a static data member shared by all the objects of the class.

Once the static data member has been defined, it exists even if no objects of its class have been created."

-- C++ standard

So static functions/variables are like **global variables**, except they exist (and are accessed) via a **scope**

Parsers.h

Designed so that any function that loads external data should go here. (Note passing data by reference)

```
class Parsers {
    static TGAInfo* loadTGA(std::string filename);
public:
    static bool parseOBJ(std::string filename,
        std::vector<float>& vertices,
        std::vector<float>& uvs,
        std::vector<float>& normals,
        std::vector<unsigned int>& indices);
    static GLint parseTexture(std::string filename);

};
```

exception is the Shader class - which we will develop later on

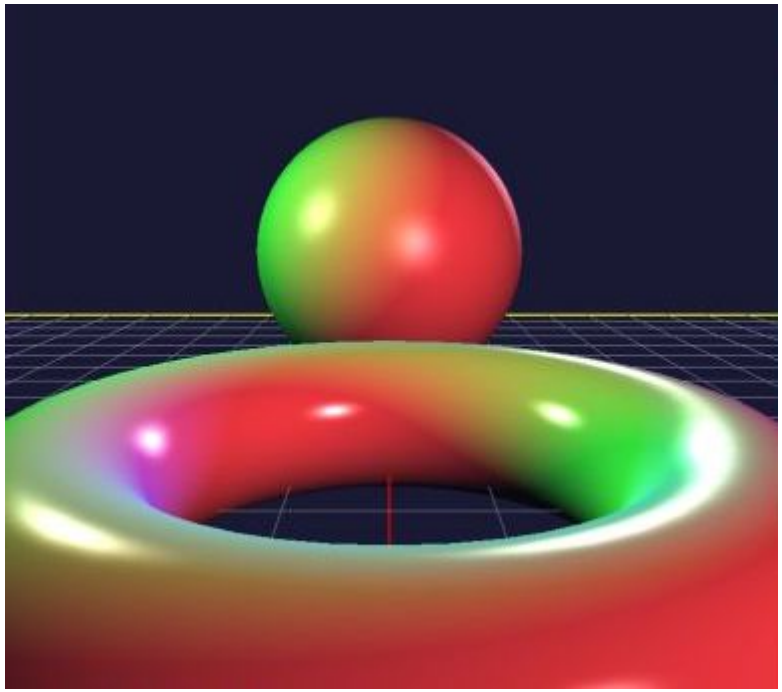
Look at usage of parsers

...in Game.cpp and GraphicsSystem.cpp

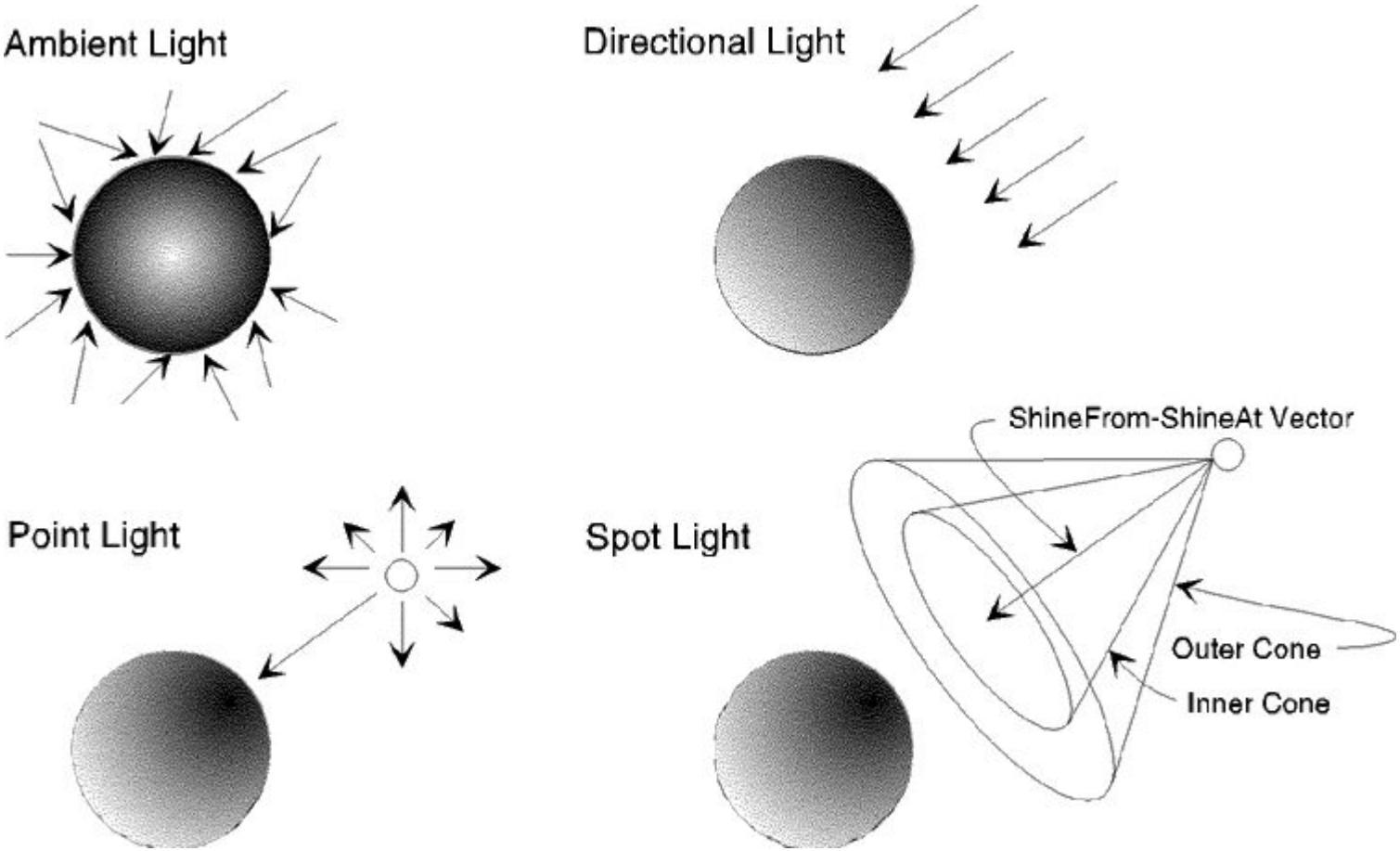
Light Component

By creating a light component we are adding the possibility of adding multiple lights to our scene.

What should the properties of a light component be?



Light Types


Ambient Light
Directional Light
Point Light
Spot Light
ShineFrom-ShineAt Vector
Outer Cone
Inner Cone

Point light Component

Only needs color property.

Why not position?

Shader for multiple lights

```
struct PointLight {
    vec3 position;
    vec3 color;
};
```

We can create a struct in GLSL as a container for data. We will fill a uniform of this struct type with data from our engine.

Arrays of uniforms

```
const int MAX_LIGHTS = 8;
uniform PointLight lights[MAX_LIGHTS];
uniform int u_num_lights;
```

There is no such thing as dynamic memory for shaders, arrays must be hard coded to a certain size.

However, we can simulate a dynamic array size using a uniform **u_num_lights** (which we can change at runtime). We can then iterate over our lights as follows:

```
//loop lights
for (int i = 0; i < u_num_lights; i++){
    //calculate illumination
}
```

Shader optimisation

In the old days, it was considered bad to implement:

- Conditional statements (e.g. *if*)
- Static loops
- Dynamic loops
- etc.

In your shader. Why?

Because the multiple cores of the GPU ***invoke operations in parallel*** - the only thing that changes are the attributes.

This is usually called the ***wavefront***.

Wavefront divergence

There are three types of possible divergence

1. Static branching: resolved at compile time, same for all invocations of wavefront, always optimised, never breaks wavefront

2. Static uniform branch: condition depends on uniform. As uniform is same for all invocations, rarely a problem.

3. Dynamic branch: based on some condition calculated in shader. Possible wavefront divergence.

Modern GPUs handles divergence better

Desktop, pre 2007: “never use if statements unless static”. Some hardware is okay, some not.

Desktop, 2007-2010: static/uniform branching okay, dynamic maybe not.

Desktop, post 2010: dynamic branching is fine.

Mobile GPUs

OpenGL ES 2 (pre 2013ish): pretty bad, “never use if”.

ES 3 (post 2013ish). Static branching is fine, some hardware deals okay with dynamic branching

Take home message

A for loop of lights using a uniform condition is going to be well optimized.

Dynamic branching is going be essential later on. Don't worry about it now, optimisation will come later.

Accessing GLSL arrays in OpenGL

Simple use a const char* of the variable name in GLSL, and use standard opengl calls for uniforms

```
std::string light_position_name = "lights[0].position"; // uniform name
GLint u_light_pos = glGetUniformLocation(current_program_, light_position_name.c_str());
if (u_light_pos != -1) glUniform3fv(u_light_pos, 1, lm::vec3(1000.0,0.0,1000.0).value_);
```

If we were iterating lights in a 'for' loop, how could we access light 'i'?

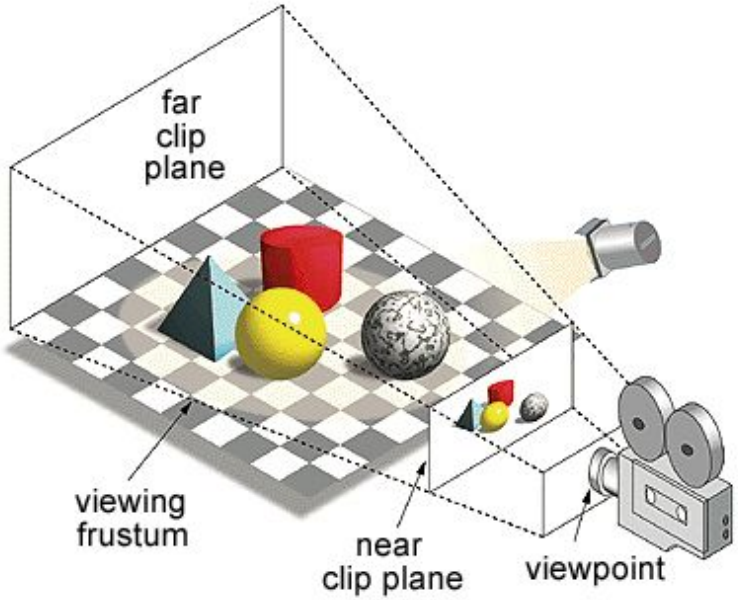
Camera component

What are the basic matrices that define a camera?

What info do we need to define them?

A camera component allows us to define multiple cameras in a scene

Do we need to do that?



Camera Component properties

Vectors and floats used to set matrices

View, Projection, and ViewProjection matrices

functions to set/update matrices

NOTE: camera position is required to update view matrix - but it makes **Transform component redundant** - need to be careful with this...

Today

1. Implement Light component

2. Modify shader to accept multiple lights

3. Implement Camera Component

4. Create two entities with cameras, switch between them every *x* frames