Game Engine Programming
Lab 10: Multiple Cameras

Karsten Pedersen
Department of Creative Technology

October 6, 2019

In this lab we will look into implementing multiple cameras into the engine in order to open up new functionality such as split screen local multiplayer, render textures and post processing.
Lets begin by looking at our existing code:

```
while(running)
{
  while (SDL_PollEvent(&event))
  {
    if(event.type == SDL_QUIT)
    {
      running = false;
    }
  }

  getWorld()->tick();
  getWorld()->display();
}
```

This allows the game loop to function and each frame we update the world in *tick* and then draw the world via *display*. We will focus on the rendering of the world. In your *MeshRenderer* class or alternative you should have something similar to:

```
std::shared_ptr<Entity> ce = getWorld()->getEntity<Camera>();

shader->setModel(getTransform()->getModelMatrix());
shader->setView(glm::inverse(ce->getTransform()->getModelMatrix());
shader->setProjection(ce->getComponent<Camera>()->getProjection());
```

This makes the assumption that there is a single camera in the world and in order to retrieve it we scan the entities for one with that component attached. Instead lets store the *Current* camera as a reference whilst we draw the scene once for each camera. We should change our main loop code to something similar to:

```
while(running)
{
  while (SDL_PollEvent(&event))
  {
    if(event.type == SDL_QUIT)
    {
      running = false;
    }
  }

  getWorld()->tick();
  std::vector<std::shared_ptr<Entity> > ces;
  getWorld->getEntities<Camera>(ces);

  for(size_t i = 0; i < cameras.size(); i++)
  {
    getWorld()->setCurrentCamera(cameras.at(i)->getComponent<Camera>());
    getWorld()->display();
  }
}
```

The key part here is using *setCurrentCamera* for each camera we iterate through the scene for. This means later on in the *MeshRenderer* we can retrieve it and use it for future rendering. The following listing demonstrates this:

```
shader->setView(glm::inverse(
  getWorld()->getCurrentCamera()->getTransform()->getModel()));

shader->setProjection(getWorld()->getCurrentCamera()->getProjectionMatrix());
```

Note:
With this in place; see if you can render a few instances of your scene from different locations and either toggle between them or draw them at different parts of the screen. The function *glViewport* might be useful here.

Note:
Have a look into how Unreal Engine or Unity deals with cameras and render targets. Have a read up on using OpenGL to create Frame Buffer Objects and this is what we will be looking at attaching to each camera next.