|  |  |  |
| --- | --- | --- |
|  | **Rochester Institute of Technology**  **Golisano College of Computing and Information Sciences**  **School of Interactive Games and Media**  **2145 Golisano Hall – (585) 475-7680** |  |

**Data Structures & Algorithms for Games & Simulation II**

**IGME 309, 2015 Spring**

**E08: (avoid) Gimbal Lock**

For this in-class exercise you are asked to set up a project in a particular way so then you can modify the source code and avoid the dreadful Gimbal Lock.

Create a new OpenGL solution (in the engine of your choice, but more detailed instructions will be given in case you are not using the provided Rendering Engine).

Create a glm::mat4 member variable that will hold the matrix transformation of a model (or shape) matrix4 m\_mToWorld;

Create a glm::vec3 member variable that will hold the orientation of a model (or shape).

Create a shape using your solution for homework assignment 04 (or use the m\_pMeshMngr to create a shape or load a model). Remember, if you are using your own solution, to create the variable once, if you are using ReEng you can load a model in the InitVariables method with:

m\_pMeshMngr->LoadModel("Minecraft\\MC\_Steve.obj", "Steve");

In you update method create 3 matrices that construct 3 matrix4 matrices. You will initialize said matrices with the rotations given by the angles in the member variable.

matrix4 rotX = glm::rotate(REIDENTITY, m\_v3Orientation.x, REAXISX);

matrix4 rotY = glm::rotate(REIDENTITY, m\_v3Orientation.y, REAXISY);

matrix4 rotZ = glm::rotate(REIDENTITY, m\_v3Orientation.z, REAXISZ);

Make a linear combination of these matrix by multiplying rotX, rotY and rotZ and store that new matrix in m\_mToWorld:

m\_mToWorld = rotX \* rotY \* rotZ;

Assign that matrix to your model (or shape), if you are using ReEng you can do that with the line:

m\_pMeshMngr->SetModelMatrix(m\_mToWorld, "Steve");

If you are using ReEng add the model to the render list with:

m\_pMeshMngr->AddInstanceToRenderList("Steve");

For your keyboard, make the R key reset the values of the rotation X, Y, Z to 0; if you press X, Y or Z in the keyboard the values in the vector should increase the values in the vector respectably by one and if you press the shift key while pressing these keys they should decrease. If you are using ReEng you should just add this code into the ProcessKeyboard method after the Modifiers region:

if (sf::Keyboard::isKeyPressed(sf::Keyboard::R))

{

m\_v3Orientation = vector3(0.0f);

}

if (sf::Keyboard::isKeyPressed(sf::Keyboard::X))

{

if (!bModifier) m\_v3Orientation.x += 1.0f;

else m\_v3Orientation.x -= 1.0f;

}

if (sf::Keyboard::isKeyPressed(sf::Keyboard::Y))

{

if (!bModifier) m\_v3Orientation.y += 1.0f;

else m\_v3Orientation.y -= 1.0f;

}

if (sf::Keyboard::isKeyPressed(sf::Keyboard::Z))

{

if (!bModifier) m\_v3Orientation.z += 1.0f;

else m\_v3Orientation.z -= 1.0f;

}

Run the program, you should be able to see your model rotate by pressing X, Y or Z, check that all of the rotations work and reset the transformations with R when you are done.

Restore the rotations with R and rotate the model about 90 degrees in Y then make the model revolve with Z and X. What is happening?

The phenomenon you are watching is known as a Gimbal Lock. The inclass exercise is to solve this issue by any means you could think of. You should be able to rotate the model freely AND without gimbal lock in any combination of axis, you may use quaternions or matrix multiplications.

Starup code is provided in the repository under E08\_Gimbal\_Lock

The whole inclass exercise consists in changing the code in such a way that it doesn’t have Gimbal Lock once you press X, Y or Z for your rotations.

Show the work to the TA or professor and explain what you did in order to avoid the Gimbal Lock.