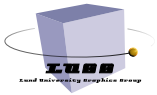
Hierarchical Transformation

EDAF80: Computer Graphics

Rikard Olajos



LUND UNIVERSITY
LTH
FACULTY OF ENGINEERING

AGENDA

1. Hierarchical Transformation and Scene Graphs
2. C++/OpenGL Framework
3. Visual Studio
4. Assignment 1

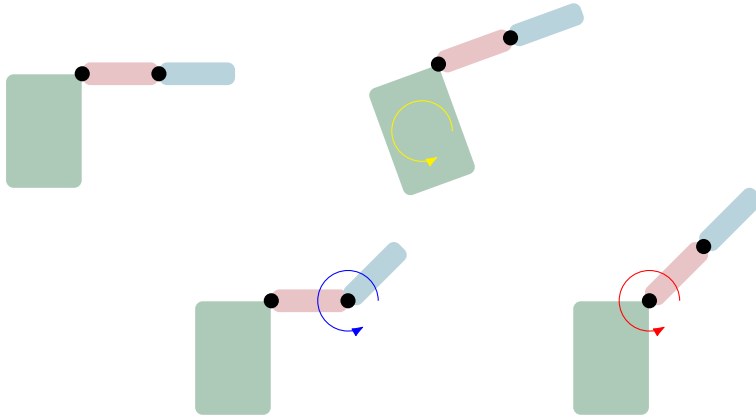
FROM LECTURE

$$T = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

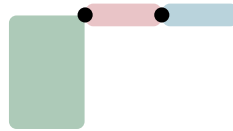
$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$S = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

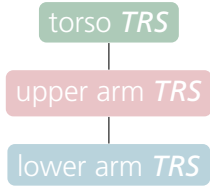
RIGID BODY



RIGID BODY



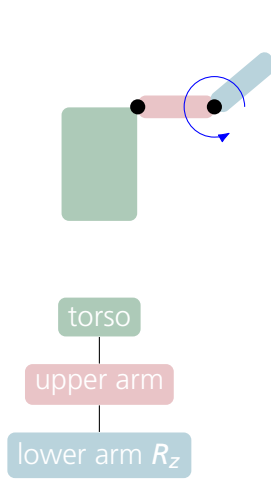
Geometry



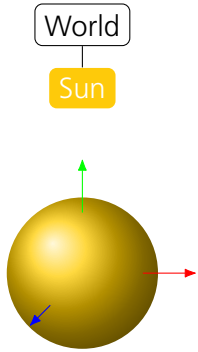
Scene graph representation

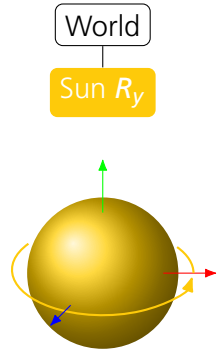
Transformations applied during rendering

RIGID BODY

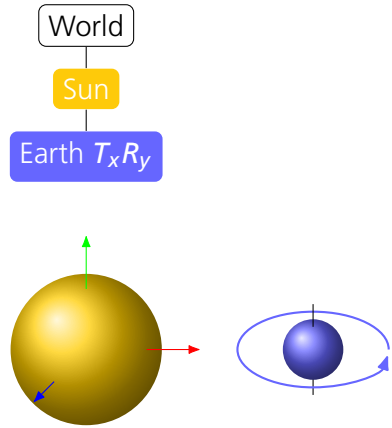


SPIN

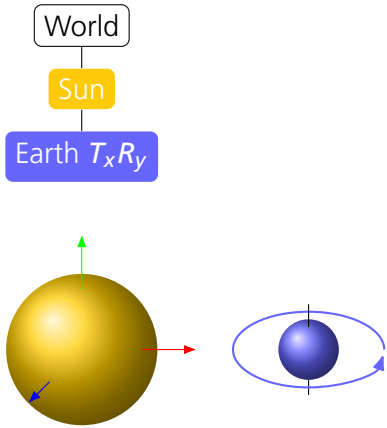


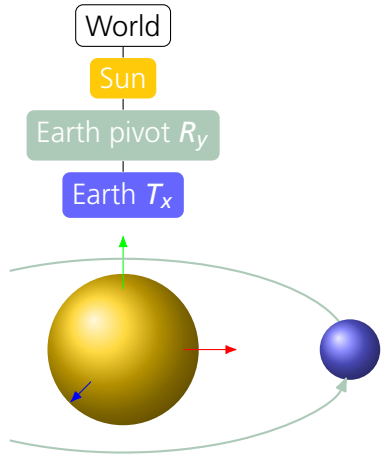


SPIN

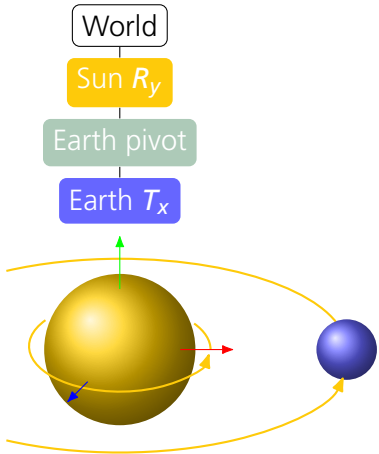


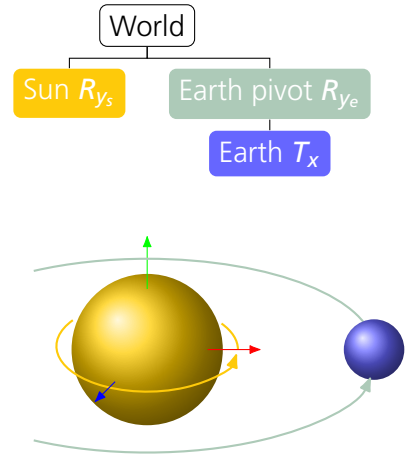
ORBIT



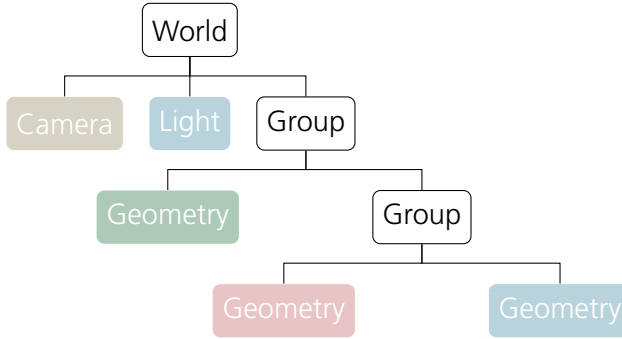


UNCOUPLING SPIN AND ORBIT





SCENE GRAPHS



C++/OPENGL FRAMEWORK: LIBRARIES

- C++/OpenGL framework: **Bonobo**
- User interface: **GLFW**, **imgui**
  - Window, GL context, mouse, key, log window, GUI
  - Using OpenGL 4.1
- Resource loading
  - Model/geometry loading: **assimp**
  - Image/texture loading: **stb**
- Vector algebra library: **GLM**
  - Based on OpenGL Shading Language (GLSL) specification
- Don’t need to look at this code, just use them as tools

C++/OPENGL FRAMEWORK: FUNCTIONS

- Mesh structure (`bonobo::mesh_data`)
- Node class
  - Child pointers to build a simple scene graph
  - `render(proj, trans)`
    - Member function to draw node
    - Takes two matrices: projection and transformation
- OpenGL texture setup function (`loadTexture2D()`)
- Shader setup: loading, compiling, linking (`createProgram()`)
- `while` loop to render scene graph
  - Add per frame node operations here (for example: `sun.rotate_y(0.01f)`)
  - Pushes `root_node` onto stack, then process all child nodes

CREATING A NODE

- `sphere` and `shader` are set up and can be used as is

```
Node sun = Node();
sun.set_geometry(sphere);
sun.set_program(shader);

GLuint sun_texture = loadTexture2D("sunmap.png");
sun.add_texture("diffuse_texture", sun_texture, GL_TEXTURE_2D);
```

ADDING MORE NODES

- Add more nodes and start building the scene graph

```
Node world = Node();

world.add_child(&sun);
...
sun.add_child(/* Add planets */);
```

MOVING NODES

- Use translation, rotation and scaling functions
- Use a time variable to animate

```
/* Absolute transformations */
sun.get_transform().SetScale(2.0f);
earth.get_transform().SetTranslate(glm::vec3(3.0f, 0.0f, 0.0f));
earth.get_transform().SetTranslate(glm::vec3(time, 0.0f, 0.0f));

/* Relative transformations */
earth.get_transform().Scale(0.9f);
sun.get_transform().RotateY(0.7f);

/* Useful rotation transformations */
earth.get_transform().LookTowards(...); /* Look in a given direction */
earth.get_transform().LookAt(...);      /* Look at a fixed point */
```

INTERACTION

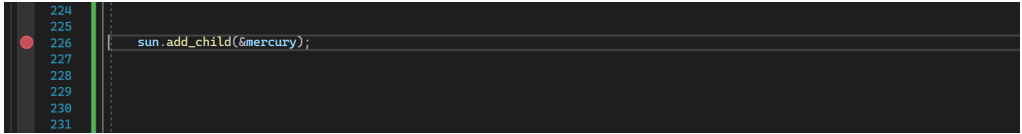
- FPS camera
  - Keyboard: "WASD" to move camera forward/left/backward/right
  - Keyboard: "QE" to move up/down
  - `shift` and `ctrl` to modify speed of movement
  - Mouse: Click and drag left mouse button
- User interface (imgui) with mouse

VISUAL STUDIO DEBUGGING

- Breakpoints
- DataTips
- printf

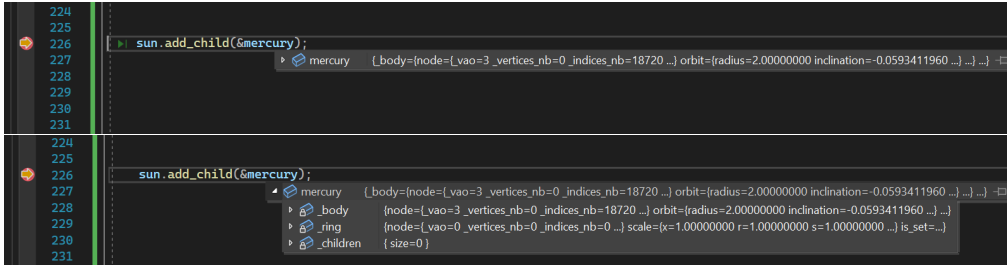
BREAKPOINTS

- A breakpoint pauses execution
- Allows for inspection of variables, stepping of lines
- Toggle breakpoint on currently line with **F9**, or click the area to the left of the line
- Right-click a breakpoint to add conditions
- Once breakpoint is hit, step (**F10**), step into (**F11**) or continue (**F5**) execution



DATATIPS

- Inspect/edit variables by hovering above them with mouse pointer
- Click to expand
- Right-click and select “Watch” to pin variable to the Watch-window



printf

- Brute force debugging
- Print whatever you need to monitor to the standard output (console window)

```
printf("Monitored value: %f\n", var);
```

- Or use `std::cout`

```
std::cout << "Monitored value: " << var << "\n";
```

- Can format the output
- Can monitor output continuously as the program executes
- Messy code ☹

ASSIGNMENT 1

- **Model the solar system!**
  - Sun, planets, moons, comets, ...spaceships? It's up to you
  - Resources included in the Bonobo framework are at your disposal
  - Models, textures, shaders
  - Code for how to add shaders us included (more in assignment 3–4)
- See assignment description for details
  - Available on the course webpage
  - Source code in assignment description
- Files you have to modify
  - `src/EDAF80/assignment1.cpp`
  - `src/EDAF80/CelestialBody.cpp`

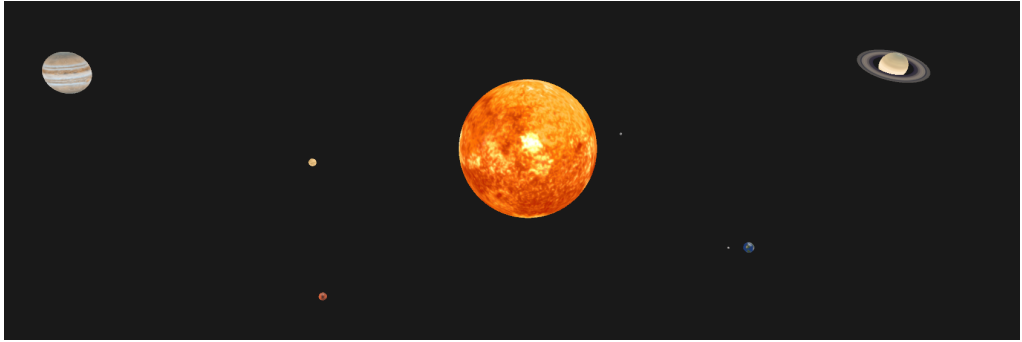
CELESTIAL BODY

```
class CelestialBody
{
public:
    CelestialBody(...);
    glm::mat4 render(...);
    void add_child(CelestialBody* child);
    ...

private:
    struct {
        Node node;
        struct { /* radius, inclination, speed, rotation_angle */ } orbit;
        struct { /* axial_tilt, speed, rotation_angle */ } spin;
    } _body;

    struct { Node node; ... } _ring;
};
```

RENDER

```
glm::mat4 CelestialBody::render(
    glm::mat4 const& view_projection,
    glm::mat4 const& parent_transform,
    ...)
{
    ...

    glm::mat4 world = parent_transform;

    /* Edit world matrix here */
    auto const scale = glm::scale(glm::mat4(1.0f), glm::vec3(0.5f));
    world = world * scale;

    /* Supply full transformation matrix! */
    _body.node.render(view_projection, world);

    return parent_transform;
}
```

DEMO



NEXT

- Download the code and get started
- Post questions on the discussion forum