A3: Moving Planets

2017313135 소프트웨어학과 권동민

Data Structure

I started this project with "trackball", and so I used more file(circle.h, circ.frag, circ.vert -> copy from last project, fixed them)

In Circle_t structure, I added two variables "id" and "global_theta" for each rotation and orbit. Id is used to identify each planet, and global_theta is used to calculate orbit angle. And I used buffer queue for store spheres to draw them into screen.

Algorithm

Planets' orbit position

I assume that all Planets orbits only in xy-planet.(not in opengl coordinate system) And I set the variable "global_theta" that is angle between x-axis and the line(connecting the sun and the planet) in xy-planets. if the planet's id is 0("0" is Sun's id), then I didn't add constant value to its global theta. But if not, all planet has their own constant value that is added to their "global_theta" variable. Each constant value is arbitrarily determined by me. And finally I set the each planet's position by using this variable. It can be possible by set the translate matrix in update(in circle_t structure) function. The matrix can be shown as follows.

$$\begin{bmatrix} 1 & 0 & 0 & center.y * -\sin(global\_theta) \\ 0 & 1 & 0 & center.y * \cos(global\_theta) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

It is just multiply -sin() to x-value, and cos() to y-value. That's all.

Synchronization spheres' rotation speeds across different computer

In update function, I added constant value to each planet's "theta" and "global_theta" value to change the position of the planets. But if I just added constant value, the rotation speed is different over computer. Thus, the rotation speed should increase in proportion to time. So I calculated the execution time and multiply with constant value. For calculate execution time(time stamp) I set the bt(the time of previous frame) and t(the time of current frame) variables and calculate difference between them.

$$theta = theta + c_{theta} * (t - bt) * K \ (k\ is\ constant\ value\ for\ normalize)$$

$$global\_theta = global\_theta + c_{global\_theta} * (t - bt) * K$$

Zooming and Panning

First I set Key flags “shift”, “ctrl” variables. when I press the shift button, the “shift” value will become true, and if I release the button, then false. It is same with “ctrl”. So if I click the mouse’s left button, I’ll check the flags and determine what current operation(Zooming or Panning or trackball) it is.

When I click mouse, then set initial position of mouse cursor. And in motion function, I calculate the difference between current position of cursor and the initial value. It is same in both operations. Zooming can be implemented simply. I just changed the “eye” value in view matrix.(the value is placed in _34 position in view matrix) so I just added the calculated value(current position – initial position) to initial view_matrix(this matrix is also set in click function). It is almost same in Panning. When the mouse-left button clicked, I set initial panning position. And if the mouse move, I calculate (current position-initial position of cursor) and add to initial panning position. Finally I add panning position vector to translate matrix.