Computer Graphics

Homework 4

Interactive 3D Application

2015-11-26 ~ 12-16

2012726090

조성재

Cho Sung Jae

Contents

Necessary features

Necessary features	Details
3D modeling	• Spheres by traingle strips
Viewing	• 9 different viewpoints • The camera is always toward the current viewing object. • The camera revolutes with the current viewing object for natural perception.

Lighting	• 2 distant lights
	  o The first light: toward (0, 1, 0) direction
	  o The second light: toward (0, -1, 0) direction

Texture mapping	• Texture mapping to spheres o 9 different texture mapping • 1 white plain texture for text rasterization
Multiple independent objects	• 10 objects: Sun, Mercury, Venus, Earth, Mars, Jupiter, Saturn, Uranus, Neptune, Moon • Independent movement: rotation, tilt, and revolution.
Animation	• Rotation • Revolution • Different speed of each rotations and revolutions based on the real solar system.

Interaction	• Keyboard
	  o Camera movement using 'd', 'D', '→', '←', '↑', and '↓'.
	• Mouse
	  o Change the viewpoint among the 9 viewpoints: Sun, Mercury, …, Moon.
	  o Select real distance mode or close distance mode

Text output	• A name tag is attached to each object.

Main concepts

I made a program simulate the solar system.

Objects

- I made 10 astronomical objects: Sun, Mercury, Venus, Earth, Mars, Jupiter, Saturn, Uranus, Neptune, and Moon.
- Each object rotates itself and some objects have their own object to revolute(공전).

Astronomical objects	Rotate	Revolute
Sun	Yes	None
Mercury	Yes	Sun
Venus	Yes	Sun
Earth	Yes	Sun
Mars	Yes	Sun
Jupiter	yes	Sun
Saturn	Yes	Sun
Uranus	Yes	Sun
Neptune	Yes	Sun
Moon	Yes	Earth

- I implemented rotating speeding and revolution speed based on the real solar system.

Implemetation details

Drawing the Scene

Drawing the Scene

```
void drawScene()
{
	glMatrixMode(GL_MODELVIEW);

	glPushMatrix();
	{
		drawSphere(SUN);
		drawSphere(MERCURY);
		drawSphere(VENUS);
		drawSphere(EARTH);
		drawSphere(MOON);
		drawSphere(MARS);
		drawSphere(JUPITER);
		drawSphere(SATURN);
		drawSphere(URANUS);
		drawSphere(NEPTUNE);
		drawSphere(MOON);

	}
	glPopMatrix();
}
```

Drawing spheres

- Drawing by triangle strips
- Translation and rotate functions for rotation and revolution.

Drawing spheres

```
void drawSphere(SolarSystem elementIndex)
{
	AstronomicalObject & ao = getAstronomicalObject(elementIndex);
	GLfloat theta = 0;
	GLfloat phi = 0;
	GLfloat radius = ao.getRadius();
	GLint slice = 36;
	GLint stack = 18;
	const GLfloat delta_theta = (360 / slice) * DEGREE; // the increment of 
theta(radian)
	const GLfloat delta_phi = (180 / stack) * DEGREE; // the increment of phi(radian)

	glBindTexture(GL_TEXTURE_2D, texID[numTextures-1]);
	drawFontOn(elementIndex);
	setupMaterial_silver();
	glBindTexture(GL_TEXTURE_2D, texID[elementIndex]);


	glPushMatrix();
	{
		if(&ao.getRevoluteObject() != NULL)
```

```
			glTranslatef(ao.getRevoluteObject().getX(),
ao.getRevoluteObject().getY(), ao.getRevoluteObject().getZ());
		glRotatef(ao.getAngleRevolution(), 0, 1, 0); // Revolution
		glTranslatef(0, 0, ao.getDistanceRevolution()); // Revolution
		glRotatef(ao.getAngleAxialTilt(), 0, 0, 1); // axial tilt
		glRotatef(ao.getAngleRotation(), 0, 1, 0); // Rotation
		for (theta = 0; theta < 2 * PI; theta += delta_theta)
			//for (int i = 0; i != slice; ++i, theta += delta_theta)
		{

			glBegin(GL_TRIANGLE_STRIP);
			{
				glNormal3f(0, 1, 0);
				glTexCoord2f(
					1.0 - theta / (2 * PI),
					1.0
					);
				glVertex3f(0, radius, 0);
				for (phi = delta_phi; phi < PI; phi += delta_phi)
					//for (int j = 0, phi = delta_phi; j != (stack-2); ++j,
phi += delta_phi)
				{
					glNormal3f(
						sin(phi)*cos(theta),
						cos(phi),
						sin(phi)*sin(theta)
						);
					glTexCoord2f(
						1.0 - theta / (2 * PI),
						1.0 - phi / PI
						);
					glVertex3f(
						radius*sin(phi)*cos(theta),
						radius*cos(phi),
						radius*sin(phi)*sin(theta)
						);
					glNormal3f(
						sin(phi)*cos(theta + delta_theta),
						cos(phi),
						sin(phi)*sin(theta + delta_theta)
						);
					glTexCoord2f(
						1.0 - (theta + delta_theta) / (2 * PI),
						1.0 - (phi) / PI
						);
					glVertex3f(
						radius*sin(phi)*cos(theta + delta_theta),
						radius*cos(phi),
						radius*sin(phi)*sin(theta + delta_theta)
						);
				}
				glNormal3f(0, -1, 0);
				glTexCoord2f(
					1.0 - theta / (2 * PI),
					0.0
					);
				glVertex3f(0, -radius, 0);
			}
```

```
                    glEnd();
                }
        }
        glPopMatrix();
}
```

Set up viewings

Set up viewings

```
void setupViewing(SolarSystem elementIndex)
{
        AstronomicalObject& ao = getAstronomicalObject(elementIndex);
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();


        cam_x = ao.getX() + (2 * ao.getRadius() + cam_dist) * sin(cam_phi) * sin(cam_theta
+ ao.getRadianRevolution());
        cam_y = ao.getY() + (2 * ao.getRadius() + cam_dist) * cos(cam_phi);
        cam_z = ao.getZ() + (2 * ao.getRadius() + cam_dist) * sin(cam_phi) * cos(cam_theta
+ ao.getRadianRevolution());

        gluLookAt(
                cam_x, cam_y, cam_z,
                ao.getX(), ao.getY(), ao.getZ(),
                0, 1, 0);
}
```

Animation by timer

Animation by timer

```
void timer(int timer_id)
{
        sun.increaseRotation();
        sun.increaseRevolution();
        mercury.increaseRotation();
        mercury.increaseRevolution();
        venus.increaseRotation();
        venus.increaseRevolution();
        earth.increaseRotation();
        earth.increaseRevolution();
        mars.increaseRotation();
        mars.increaseRevolution();
        jupiter.increaseRotation();
        jupiter.increaseRevolution();
        saturn.increaseRotation();
        saturn.increaseRevolution();
        uranus.increaseRotation();
        uranus.increaseRevolution();
        neptune.increaseRotation();
        neptune.increaseRevolution();
        moon.increaseRotation();
        moon.increaseRevolution();

        glutPostRedisplay();
```

```
	glutTimerFunc(time_interval, timer, 0);
}
```

AstronomicalObject Class

```
#pragma once
#include <math.h>

enum SolarSystem {
	SUN, MERCURY, VENUS, EARTH, MARS, JUPITER, SATURN, URANUS, NEPTUNE,
	MOON,
	NUM_ELEMENTS
};
class AstronomicalObject
{
public:
	AstronomicalObject(SolarSystem elementIndex, AstronomicalObject *revoluteObject);
	~AstronomicalObject();
	void setRotation(double angleRotation) { this->angleRotation = angleRotation; }
	void setRevolution(double angleRevolution) { this->angleRevolution =
angleRevolution; }
	void setRealDistanceMode(bool realDistanceMode) { this->realDistanceMode =
realDistanceMode; }
	double getRadius() { return rescaleKm(radius); }
	double getDistanceRevolution();
	AstronomicalObject& getRevoluteObject() { return *pRevoluteObject; }
	double getAngleRotation() { return angleRotation; }
	double getRadianRotation() { return degree2radian(angleRotation); }
	double getAngleRevolution() { return angleRevolution; }
	double getRadianRevolution() { return degree2radian(angleRevolution); }
	double getAngleAxialTilt() { return angleAxialTilt; }
	double getHoursOfRotation() { return hoursOfRotation; }
	double gethoursOfRevolution() { return hoursOfRevolution; }
	double getDeltaAngleRotation() { if (hoursOfRotation != 0) return (timeScale /
hoursOfRotation); else return 0; }
	double getDeltaAngleRevolution() { if (hoursOfRevolution) return (timeScale /
hoursOfRevolution); else return 0; }
	double getX();
	double getY();
	double getZ();
	void increaseRotation();
	void increaseRevolution();
private:
	double radius; // km
	double distanceRevolution; // km
	double distanceRevolutionClose;
	AstronomicalObject *pRevoluteObject;
	double hoursOfRotation; // hours
	double hoursOfRevolution; // hours
	double angleAxialTilt; // degree
	double angleRotation = 0; // degree
	double angleRevolution = 0; // degree
	double deltaRotation;
	double deltaRevolution;
	const double timeScale = 100.0; // big: fast, small: slow.
	bool realDistanceMode = false;
	// converting functions
```

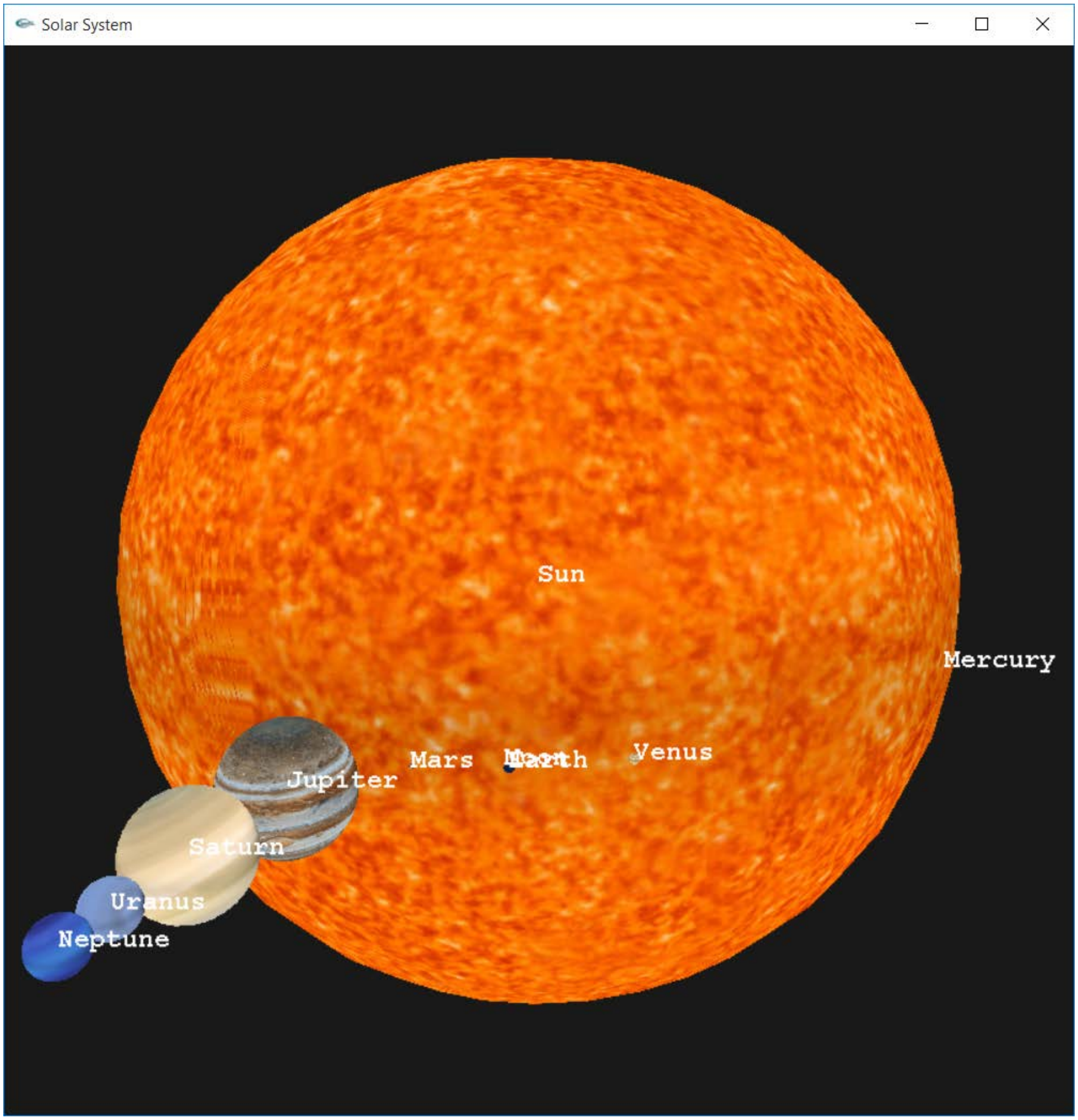
```
	double day2hour(double day) { return day * 24; }
	double year2hour(double year) { return year * 356 * 24; }
	double minute2hour(double minute) { return minute / 60.0; }
	double rescaleKm(double kilometer) { return kilometer / 6378;  } // radius of the 
earth
	const double PI = 3.141593;
	double degree2radian(double degree) { return PI / 180.0 * degree; }

};
```

Results

The First Scene

- Default viewing object: Sun
- Default distance mode: close

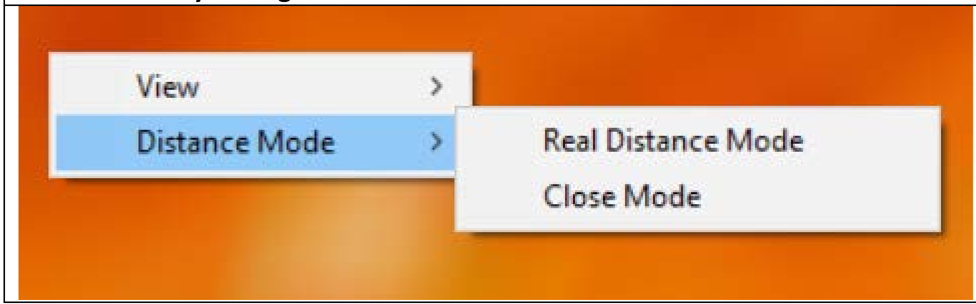


Change the Viewing Object

Select Menu by the right mouse button click

If the Earth is selected

Real Distance Mode vs Close Mode

Select Menu by the right mouse button click


Real Distance Mode	Close Distance Mode