컴퓨터그래픽스

**프로젝트결과보고서**



과 목 : 컴퓨터그래펙스

교 수 님 : 안수정 교수님

제출일자 : 2016.06.19

학 과 : 컴퓨터공학과

성 명 : 이반

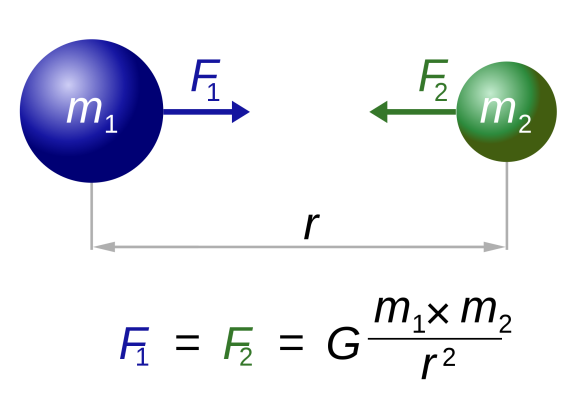
학 번 : 2014758139

**제목 : 중력 시뮬레이터 (Gravity Simulator)**

**배경**

만유인력의 법칙은 (Newton’s Law of Universal Gravitation) 입자들은 질량의 곱에 비례하지만, 또한 이들 사이의 거리의 제곱에 반비례하는 힘을 이용하여 우주의 다른 모든 입자를 끄는 것을 말한다.

아이디어는 행성을 넣고 입자를 추가하고, 입자가 이동하는 방법을 보겠다. 하나의 행성으로, 동작을 쉽게 예측할 수 있다. 그러나 식으로 다른 행성을 추가하고 상황이 혼란된다. 놀라운 패턴은 겉보기에 임의의 방법으로 나타날 수 있다.

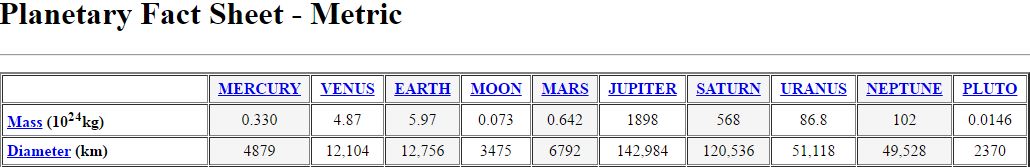


**목적**

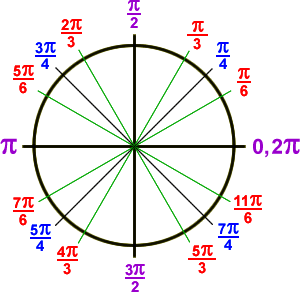
OpenGL으로 중력 시뮬레이터는 만유인력의 법칙(Newton's law of universal gravitation)에 따라 어떻게 중력의 작품을 볼 수 있는 시뮬레이션입니다. 입자가 한 행성, 또는 여러 행성 이동하는 방법을 볼 수 있다.

**서론**

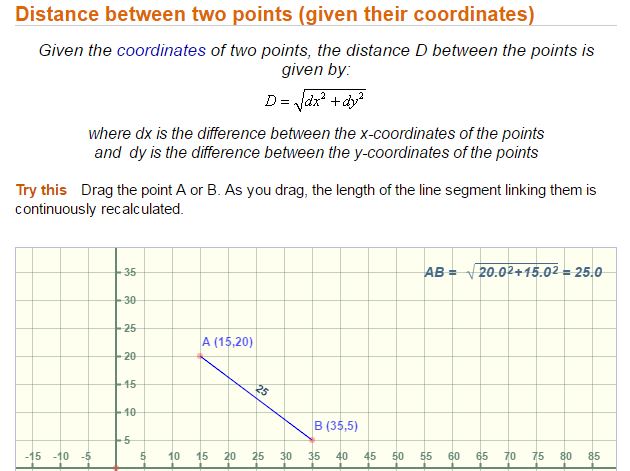
* 행성은 7개가 있다. 적용하는 행성 정보가 다음과 같다 (m=mass (질량), r=diameter (지름))
  + Earth (지구) m=5970 r=12.756
  + Venus (금성) m=4870 r=12.104
  + Mars (화성) m=642 r=6.792
  + Jupiter (목성) m=1898000 r=142.984
  + Saturn (토성) m=568000 r=120.536
  + Uranus (천왕성) m=86800 r=51.118
  + Neptune (해왕성) m=102000 r=49.528



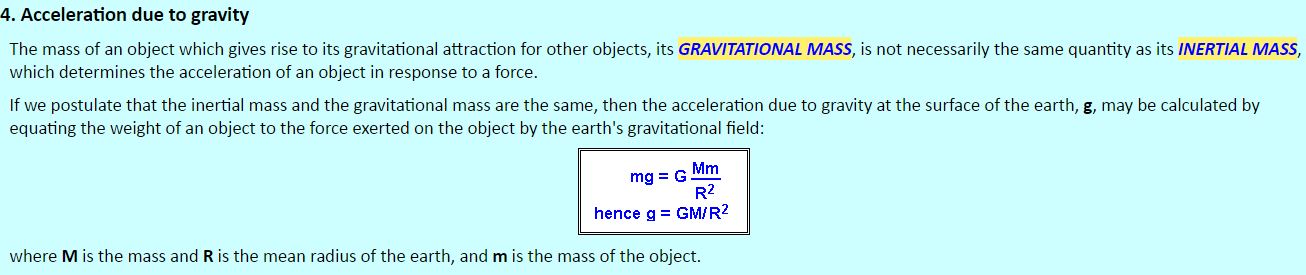
* 각 행성과 입자가 질량,지름,색상,x,y촤표,가속도에대한 정보를 저장한다.
* 행성과 입자를 그리는방법은 for문으로 원형을 그린다.



* MouseClick에따라 행성이나 입자를 그린다. Leftmouse=입자, Rigthmouse=행성, middlebutton=지우기.
* 키보드로부터 alphabet button클릭하면 선택하는 행성을 그린다. (각 행성은 식별자가 있다).
* 중력효과를 (gravitation effect) 적용하기 위해 포대기형 루프(nested loop)를 이용한다. 2개for문으로 2개 입자나 행성을 비교한다. 알고리즘은 일단 한행성과 다른 입자 사이의 거리를 계산한다. (phytagoras theorem 이용)



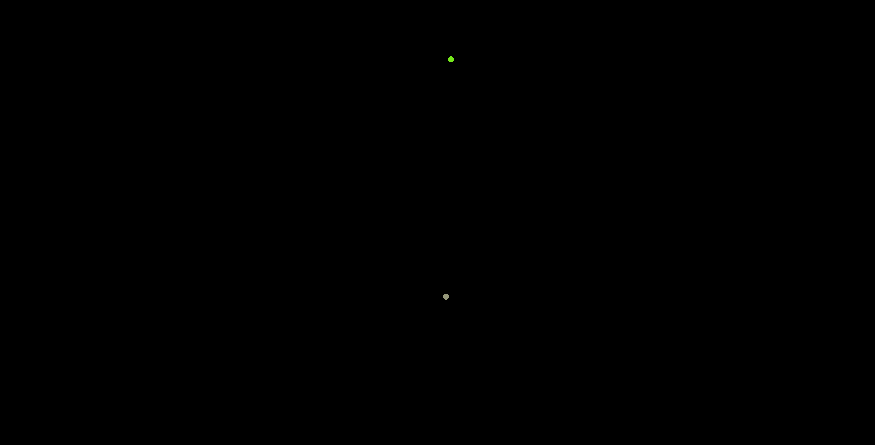
그리고 거리가 행성의 지름보다 더 작으면 입자를 지운다 (입자가 중력효과때문에 행성에 떨어지는 말이다). 아니면은 그 입자의 가속도를 계산한다 (가속도x, 가속도 y 이용). 알고리즘은 Newton’s Law of Gravitation 정리를 이용한다.



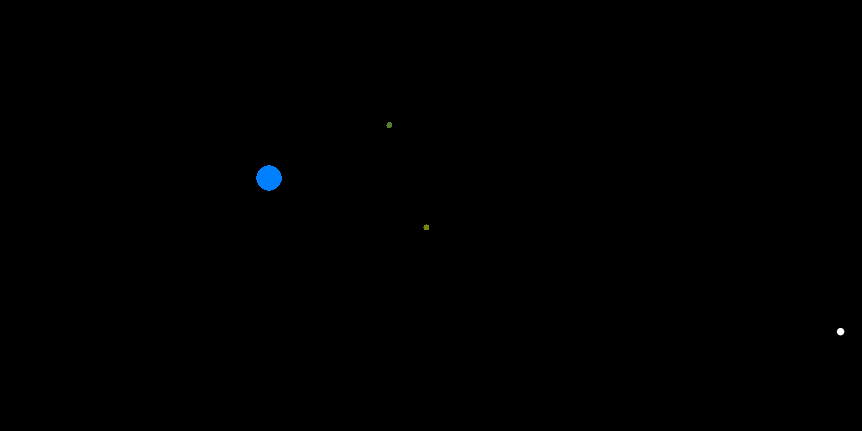
입자의 원래 x,y촤표 더하기 계산하는 x,y가속도는 입자움직을 점점 볼 수 있다 (glutTimerFunc 이용).

**수행방법 및 결과**

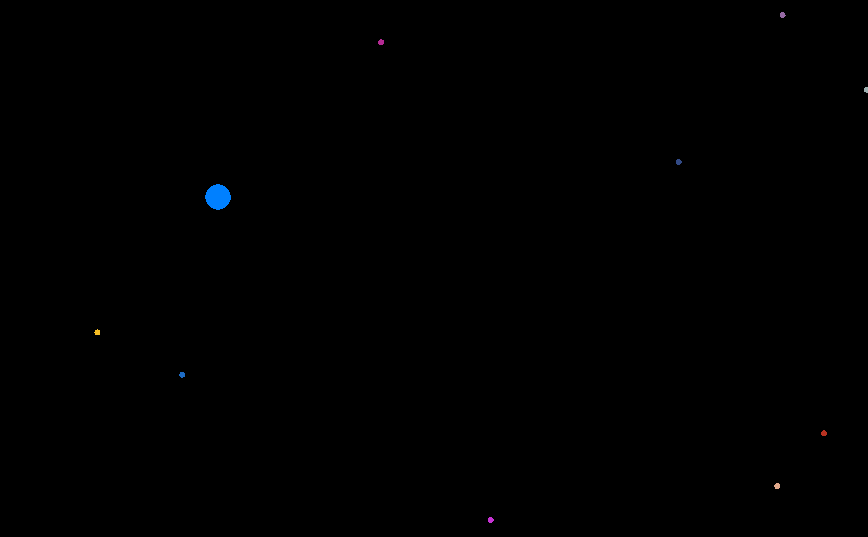
leftmouse으로 입자를 추가한다. 입자이니까 중력이 없어서 이동되지않는다.



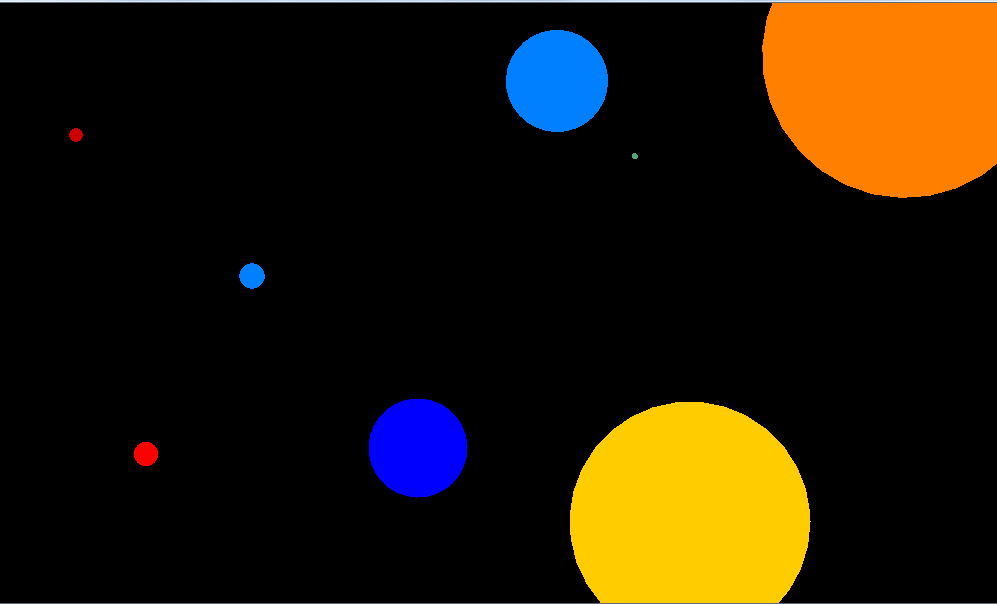
키보드로부터 ‘e’ 클릭하고 rightmouse클릭하고 지구를 추가한다. 그 입자들은 만유인력의 법칙 때문에 지구에 떨어진다.

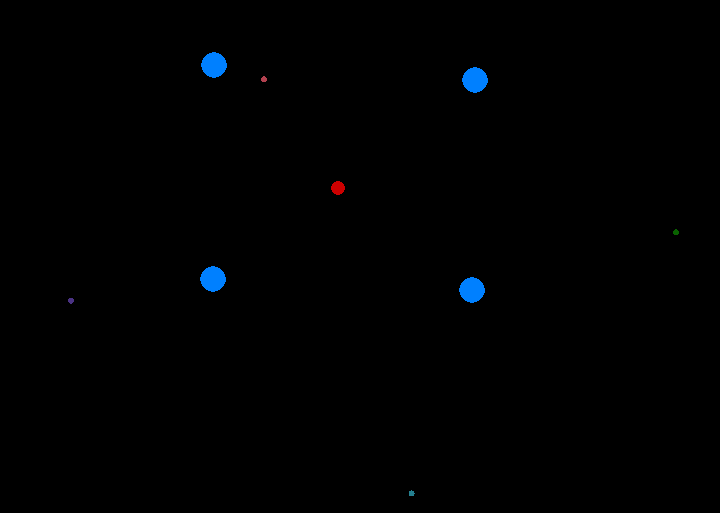


여러 가지 입자를 추가할 수 있다.



키보드로부터 각행성의 식별자로 입력해서 여러 가지 행성을 추가한다. 입자들이 어디에 떨어지는지 볼 수 있다. 물체를 지우고 싶으면 middlemouse클릭한다.



똑같은 행성도 추가할 수 있다.

**소스코드**

#include "stdafx.h"

#include <GL/glut.h>

#include <vector>

struct Particle {

float x;

float y;

float r; //diameter 지름

float ax; //acceleration x 가속도 x 촤표

float ay; //acceleration y 가속도 y 촤표

float m; //mass

float color[3];

};

void timer(int = 0);

void display();

void mouse(int, int, int, int);

void addParticle(float, float, bool = true, float = 0, float = 0);

void removeParticles();

void keyboard(unsigned char, int, int);

int Mx, My, WIN;

bool PRESSED\_LEFT = false, PRESSED\_RIGHT = false,

PRESSED\_MIDDLE = false, SPEED\_PARTICLES = false,

earth=false, venus=false, mars=false,jupiter=false,uranus=false,saturn=false,neptune=false;

std::vector<Particle> particles;

int main(int argc, char \*\*argv)

{

glutInit(&argc, argv); //initialize glut library

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB); //sets the initial display mode

glutInitWindowSize(1000, 600);

glutInitWindowPosition(50, 50);

WIN = glutCreateWindow("Gravity Simulator");

glClearColor(0, 0, 0, 1);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity(); // replace the current matrix with the identity matrix

glOrtho(-500.0, 500.0, 300.0, -300.0, 0, 1); //multiply the current matrix with an orthographic matrix

glutDisplayFunc(display);

glutMouseFunc(mouse);

glutKeyboardFunc(keyboard);

timer();

glutMainLoop();

return 0;

}

void timer(int)

{

display();

if (PRESSED\_LEFT && !SPEED\_PARTICLES)

{

addParticle(10, 3); //입자 추가

PRESSED\_LEFT = false;

}

if (PRESSED\_RIGHT)

{

if (earth)

{

addParticle(5970, 12.756, 0); //지구 추가

earth = false;

}

if (venus)

{

addParticle(4870, 12.104, 0); //금성 추가

venus = false;

}

if (mars)

{

addParticle(642, 6.792, 0); //화성 추가

mars = false;

}

if (jupiter)

{

addParticle(1898000, 142.984, 0); //목성 추가

jupiter = false;

}

if (saturn)

{

addParticle(568000, 120.536, 0); //토성 추가

saturn = false;

}

if (uranus)

{

addParticle(86800, 51.118, 0); //차왕성 추가

uranus = false;

}

if (neptune)

{

addParticle(102000, 49.528, 0); //해왕성 추가

neptune = false;

}

PRESSED\_RIGHT = false;

}

if (PRESSED\_MIDDLE)

removeParticles(); //모든 입자와 행성을 지운다

for (int i = 0; i < particles.size(); i++) //첫 입자/행성

{

Particle &p = particles[i];

bool not\_fall = true;

for (int j = 0; j < particles.size(); j++) //다음 입자/행성

{

if (j == i || p.m >= 100) // 행성이면 이동되지 않다

continue;

const Particle &p1 = particles[j];

float d = sqrt((p1.x - p.x)\*(p1.x - p.x) + (p1.y - p.y)\*(p1.y - p.y)); //2object 사이의거리 계산=피타고라스

if (d > p1.r) //거리 > p1의 지름

{ //Newton's Acceleration due to gravity theorem 만유인력의 법칙으로 계산

//ax=G\*m/D^2 \* cosx , ay=G\*m/D^2 \* siny

p.ax += 0.0667 \* p1.m / (d\*d) \* (p1.x - p.x) / d; //ax+ => acceleration keep increasing after moving (particle getting faster)

p.ay += 0.0667 \* p1.m / (d\*d) \* (p1.y - p.y) / d; //이동한 후에 가속도 증가

//0.0667=gravitational constant, 원래값 =6.67\*10^-11

}

else

not\_fall = false; //fall=true, 입자를 지운다

}

if (not\_fall)

{

p.x += p.ax; //set particle coordinate + coordinate from ax and ay

p.y += p.ay; //입자 촤표계 + 가속도촤표계

}

else

particles.erase(particles.begin() + i); //delete when falls to the big particle 입자 지운다

}

glutTimerFunc(1, timer, 0); //callback, 저장하는 촤표계로 display() 수정

}

void display()

{

glClear(GL\_COLOR\_BUFFER\_BIT);

//draw particles

for (int i = 0; i < particles.size(); i++)

{

Particle &p = particles[i]; //get particle data from vector

glColor3f(p.color[0], p.color[1], p.color[2]);

glBegin(GL\_POLYGON);

for (float c = 0; c < 2 \* 3.14; c += 0.2) //full circle degree, 2\*pi

glVertex2f(p.r\*cos(c) + p.x, p.r\*sin(c) + p.y); //x coordinate=r\*cos a+saved x coordinate, so does y coordinate

glEnd();

}

glFlush();

glutSwapBuffers();

}

void mouse(int button, int state, int x, int y)

{

//set the coordinates (x,y촤표 설정)

Mx = x - 500; //position of x particle after clicked (make it center)

My = y - 300;

//check which button is pressed, 어느 클릭돤 버튼을 확인한다

if (button == GLUT\_LEFT\_BUTTON)

PRESSED\_LEFT = state == GLUT\_DOWN;

else if (button == GLUT\_RIGHT\_BUTTON)

PRESSED\_RIGHT = state == GLUT\_DOWN;

else if (button == GLUT\_MIDDLE\_BUTTON)

PRESSED\_MIDDLE = state == GLUT\_DOWN;

}

void addParticle(float m, float r, bool randColor, float ax, float ay)

{

Particle p;

p.x = Mx; //입자 위치=mouse 위치

p.y = My;

p.ax = ax / 30;

p.ay = ay / 30; // slow down the speed a little

p.m = m;

p.r = r;

if (randColor) //입자, 색상변하

{

p.color[0] = rand() % 200 / 200.0;

p.color[1] = rand() % 200 / 200.0;

p.color[2] = rand() % 200 / 200.0;

}

else if (earth) //행성의 색상 설정

{

p.color[0] = 0;

p.color[1] = 0.5;

p.color[2] = 1;

}

else if (venus)

{

p.color[0] = 1;

p.color[1] = 0;

p.color[2] = 0;

}

else if (mars)

{

p.color[0] = 0.8;

p.color[1] = 0;

p.color[2] = 0;

}

else if (jupiter)

{

p.color[0] = 1;

p.color[1] = 0.5;

p.color[2] = 0;

}

else if (saturn)

{

p.color[0] = 1;

p.color[1] = 0.8;

p.color[2] = 0;

}

else if (uranus)

{

p.color[0] = 0;

p.color[1] = 0.5;

p.color[2] = 1;

}

else if (neptune)

{

p.color[0] = 0;

p.color[1] = 0;

p.color[2] = 1;

}

particles.push\_back(p);

}

void removeParticles()

{

for (int i = 0; i < particles.size(); i++)

particles.pop\_back(); //vector에서 입자/행성을 제거

}

void keyboard(unsigned char key, int x, int y)

{

switch (key) //행성 식별자, 키보드로부터 입력

{

case 'e':

earth = true;

break;

case 'v':

venus = true;

break;

case'm':

mars = true;

break;

case'j':

jupiter = true;

break;

case's':

saturn = true;

break;

case'u':

uranus = true;

break;

case'n':

neptune = true;

break;

case 27: //esc

removeParticles();

glutDestroyWindow(WIN);

exit(0);

break;

}

}

**결론**  
이 간단한 시뮬레이터로 우리는 중력이 지구와 입자를 통해 어떻게 작동하는지 볼 수 있다. 만유 인력의 뉴턴의 법칙으로 입자의 가속도를 계산할 수 있다. 하나의 입자가 중력를 통해 여러 행성에 어떻게 반응하는지 볼 수 있다. 입자가 다른 행성에 한 행성에서 방향을 변경하는 방법을 볼 수 있다. 하지만이 시뮬레이션은 여전히 많은 개선이 필요한다. 우리는 스타, 궤도를 추가하고, 행성의 색상 그라데이션을 적용 할 수 있다.

비고

https://en.wikipedia.org/wiki/Newton%27s\_law\_of\_universal\_gravitation

http://www.testtubegames.com/gravity.html

http://stackoverflow.com/questions/5510230/example-of-point-based-gravity-in-c

https://www.youtube.com/watch?v=TXY6NJm5se0