KITECH

ODE Robotics Example

ODE와 OpenGL을 사용한 로보틱스 시뮬레이션

하승석, 양광웅

본 프로그램은 ODE와 OpenGL을 사용하여 로봇을 동역학 환경에서 쉽게 시뮬레이션 할 수 있도록 구성한 예제 프로그램 이다.

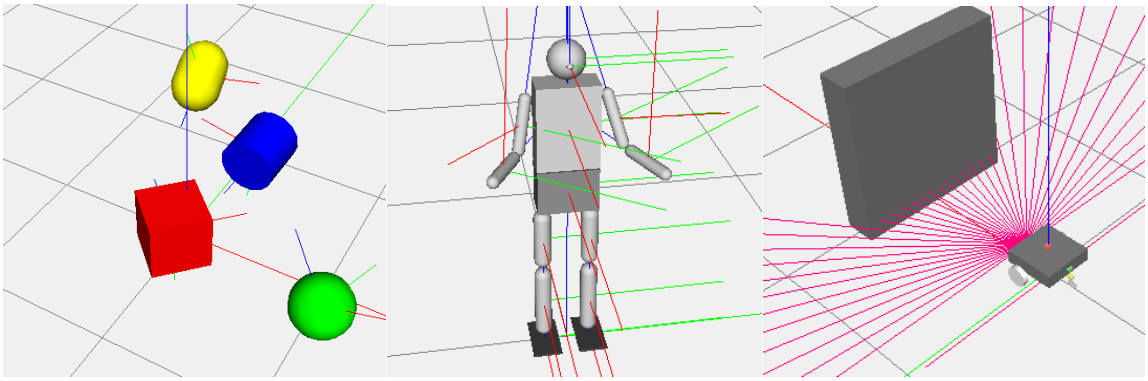
1 소개

로봇의 개발은 시간과 비용이 많이 소요되는 작업이다. 특히 연구를 목적으로 하는 고사양의 로봇은 기구부 제작 비용이 높아 여러 대를 만들기 어렵다. 그래서 여러 개발자가 동시에 로봇 제어 소프트웨어를 개발하는 것이 어렵고 사전에 충분한 테스트를 할 수 없게 된다. 하지만 로봇 시뮬레이터의 사용은 시뮬레이션 로봇을 여러 개발자가 동시에 활용하여 개발 가능하고 개발된 소프트웨어를 시뮬레이션 환경에서 쉽게 테스트 할 수 있다.

본 예제 프로그램들은 ODE(Open Dynamics Engine)를 사용하여 로봇을 동역학 환경에서 쉽게 시뮬레이션 할 수 있도록 한다. 로봇에 특화된 C++ API들 제공하여 시뮬레이션 프로그램을 쉽게 작성할 수 있도록 하며, 간단한 박스/실린더/구/캡슐 등의 도형과 구상/회전/직선/고정 관절, 거리/힘/토크/가속도 센서를 제공하여 단순한 바퀴형 이동로봇에서 복잡한 휴머노이드형 로봇까지 시뮬레이션 모델을 쉽게 구축할 수 있도록 한다.

ODE는 관절로 연결된 강체 동역학을 시뮬레이션 하는 공개 라이브러리다. 따라서 차량이나 가상 현실 환경에서 움직이는 물체들을 시뮬레이션 하는데 적합하다. Russell Smith가 ODE 개발을 시작했고 현재 몇 명의 공헌자들의 도움을 받고 있다.

* ODE 홈페이지: http://www.ode.org/



ODE Examples

2 프로그램 실행 환경

다음은 ODE & OpenGL를 사용한 예제 프로그램을 컴파일 하고 실행하기 위한 PC 환경이다.

System Requirements:

OS	Windows XP, 7
Compiler	Microsoft Visual C++ 2008 SP1
Library	ODE 0.11.1

3 ODE Wrapper class

다음 설명은 ODE Wrapper class(Ode.h, Ode.cpp)의 중요한 함수에 대한 설명이다.

3.1 시뮬레이션 진행

```
void Step (double dt);
map<string, sGeometry *> *pGetOdeGeomList ();
```

Step

시뮬레이션을 한 스탭 진행한다.

- double dt – 시간 간격 (단위: sec)

pGetOdeGeomList

물체 목록을 얻어온다. 주로 OpenGL로 물체를 그리기 위해 사용한다.

3.2 물체 추가

Box, Sphere, Cylinder, Capsule, Ray와 같은 물체를 생성한다.

```
void AddBox     (const char *name, double mass, COLORREF color,
   double x, double y, double z, double roll, double pitch, double yaw,
   double dx, double dy, double dz);
void AddSphere  (const char *name, double mass, COLORREF color,
   double x, double y, double z, double roll, double pitch, double yaw,
   double radius);
void AddCylinder(const char *name, double mass, COLORREF color,
   double x, double y, double z, double roll, double pitch, double yaw,
   double radius, double height);
void AddCapsule (const char *name, double mass, COLORREF color,
   double x, double y, double z, double roll, double pitch, double yaw,
   double radius, double height);
void AddRay     (const char *name,              COLORREF color,
   double x, double y, double z, double roll, double pitch, double yaw,
   double length);
```

공통 인자:

- const char *name : 물체의 이름, 생성되는 물체의 이름은 서로 달라야 한다.
- double mass : 물체의 무게 (단위: kg), 0kg 보다 큰 무게를 설정해야 한다.
- COLORREF color : 물체의 색, RGB(red, green, blue)로 생성, red, green, blue는 0에서 255값을 가진다. Ex: RGB(255, 255, 255)
- double x, y, z: 전역 좌표계에서 물체의 초기 위치 설정 (단위: m)

- double roll, pitch, yaw: 전역 좌표에서 물체의 초기 자세 설정 (단위: rad)

AddBox
x, y, z축으로의 크기가 각각 dx, dy, dz인 박스가 공간에 추가 된다.

AddSphere
반지름이 radius인 구가 공간에 추가 된다.

AddCylinder
원통의 반지름이 radius이고 원통의 높이가 height인 원통이 공간에 추가된다.

AddCapsule
캡슐의 반지름이 radius이고 캡슐의 원통부분 높이가 height인 캡슐이 공간에 추가된다.

AddRay
z축 방향으로의 길이가 length인 선분이 공간에 추가된다.

Ray는 보통 거리 측정 센서의 빔으로 사용되며, 질량이 없기 때문에 중력에 의해 떨어지지 않는다. 그리고 다른 물체와 충돌하지 않도록 설정되어 있다.

Cylinder는 Cylinder와 충돌을 체크하지 않으며, Cylinder와 Capsule 간에도 충돌을 체크하지 않는다. (ODE에서 이렇게 프로그래밍 되어있다.)

3.3 두 물체를 하나의 강체로 결합

두 물체를 하나의 강체로 연결한다.

```
void SetRigidBody (const char *base_name, const char *name);
```

인자:

- const char *base_name : 합치고자 하는 대상 물체의 이름
- const char *name : 합치고자 하는 물체의 이름

3.4 관절 생성

Fixed, Hinge, Ball Joint를 생성한다.

```
void AttachFixedJoint(const char *name, const char *prev_name, const char *next_name);
void AttachHingeJoint(const char *name, const char *prev_name, const char *next_name,
    double hx, double hy, double hz, double ax, double ay, double az,
    double fMax, double lo = -dInfinity, double hi = dInfinity);
void AttachBallJoint (const char *name, const char *prev_name, const char *next_name,
    double ax,double ay,double az,
    double fMax, double lo = -dInfinity, double hi = dInfinity);
```

공통 인자:

- const char *name : joint의 이름, 생성되는 joint의 이름은 서로 달라야 한다.
- const char *prev_name: joint를 붙이고자 하는 물체의 이름
- const char *next_name : joint를 붙이고자 하는 물체의 이름
- double fMax : joint에 가해지는 최고 토크 설정 (단위: Nm)
- double lo, hi : joint의 하한 멈춤각과 상한 멈춤각 (단위: rad)

AttachFixedJoint

두 물체를 고정하는 관절을 생성한다. name을 ""(빈 문자열)로 하면 물체는 지면에 부착된다.

AttachHingeJoint

두 물체 사이에 경첩과 같이 한 방향으로만 회전하는 관절이 생성된다.

- double hx, hy, hz : joint의 회전 중심 축 방향 설정
- double ax, ay, az : joint의 회전 중심 위치 설정

AttachBallJoint

두 물체 사이에 3방향으로 자유롭게 회전 가능한 구체 관절이 생성된다.

- double ax, ay, az : joint의 회전 중심 위치 설정

만일 caster와 같은 free wheel 상태를 만들고자 할 때는 관절의 fMax 값을 0으로 설정하면 된다.

3.5 관절 속도 및 각도 설정

Joint가 회전하는 속도와 각도를 설정하고 읽어온다.

```
double GetJointAngle (const char *name, int axis = 0);
void SetJointAngle (const char *name, int axis, double pos);
void SetJointVelocity (const char *name, int axis, double vel);
```

공통 인자:

- const char *name : 속도나 위치를 설정할 관절의 이름
- int axis : 관절의 회전 축 지정, 구체 관절인 경우만 해당
  axis 값이 0 이면 x 회전축
  axis 값이 1 이면 y 회전축
  axis 값이 2 이면 z 회전축

GetJointAngle

관절의 회전 각을 읽어온다. (단위: rad)
관절의 회전각은 -π에서 π 사이 값으로 읽힌다.

SetJointAngle

관절을 pos 각도로 회전한다. (단위: rad)

- double pos : 움직이고자 하는 각도

axis 값이 0 이면 회전 범위는 -π에서 π이고,
axis 값이 1 이면 회전 범위는 –π/2에서 π/2이고,
axis 값이 2 이면 회전 범위는 -π에서 π이다.

SetJointVelocity

관절을 vel 각속도로 회전한다. (단위: rad/s)
주로 이동로봇의 좌우 바퀴의 속도를 지정하기 위해 사용한다.

- double vel : 움직이고자 하는 각속도

3.6 센서 추가

Location, Touch, Distance, Gyro, Accel, FT 센서를 생성한다.

```
void SetLocationSensor (const char *name, const char *base_name);
void SetTouchSensor (const char *name, const char *base_name);
void SetDistanceSensor (const char *name, const char *base_name, double length);
void SetGyroSensor (const char *name, const char *base_name);
void SetAccSensor (const char *name, const char *base_name);
void SetFTSensor (const char *name, const char *prev_name, const char *next_name);
```

공통 인자:

- const char *name : 센서로 사용할 물체의 이름을 지정. 물체는 이미 생성되어 있어야 한다.

- const char *base_name : 센서가 고정될 물체의 이름을 지정. 물체는 이미 생성되어 있어야 한다.

SetLocationSensor
센서가 설치된 곳의 x, y, z 위치를 측정하는 센서를 설정한다.
GPS와 같은 센서로 보면 된다.

SetTouchSensor
다른 물체와 충돌을 감지하는 접촉 센서를 설정한다.

SetDistanceSensor
다른 물체가 있는 곳까지의 거리를 측정하는 센서를 설정한다.
스캐닝 레이저 센서나 초음파 센서, IR 센서와 같은 센서로 보면 된다.

- double length : 센서의 측정 거리를 설정한다.

Distance Sensor는 항상 Ray 물체로부터 만들어야 한다.

SetGyroSensor
센서의 회전 각속도를 측정하는 센서를 설정한다.

SetAccSensor
센서에 가해지는 가속도를 측정하는 센서를 설정한다.
Acc 센서는 다른 물체와 충돌하지 않도록 설정된다.

SetFTSensor
두 물체 사이에서 작용하는 힘과 모멘트를 측정하는 센서를 설정한다.

- const char *prev_name : 센서가 고정될 물체 1
- const char *next_name : 센서가 고정될 물체 2

3.7 센서 읽기
센서에서 측정된 값을 읽어온다.

```
void GetLocation(const char *name, double &x,double &y, double &z);
bool GetTouch(const char *name);
double GetDistance(const char *name);
void GetGyroValue(const char *name, double &velX, double &velY, double &velZ);
void GetAccValue(const char *name, double &accX, double &accY, double &accZ);
void GetFTValue(const char *name, double &fx, double &fy, double &fz,
    double &tx, double &ty, double &tz);
```

GetLocation

전역 좌표계에서 센서의 x, y, z 위치를 읽어온다.

- double &x, &y, &z – x, y, z 위치 (단위: m)

GetTouch

다른 물체와의 접촉 여부를 읽어온다. (true/false 리턴)

GetDistance

다른 물체와의 거리를 읽어온다. (단위: m)

GetGyroValue

센서의 회전 각속도를 읽어온다.

- double &velX, &velY, &velZ – 센서 좌표계에서 각속도 (단위: rad/s)

GetAccValue

센서의 가속도를 읽어온다.

- double &accX, &accY, &accZ – 센서 좌표계에서 가속도 (단위: m/s^2)

GetFTValue

두 물체 사이에 작용하는 힘과 모멘트(돌림힘)를 읽어온다.

- double &fx, &fy, &fz – 센서 좌표계에서 x, y, z 축 방향 힘 (단위: N)
- double &tx, &ty, &tz – 센서 좌표계에서 x, y, z 축 방향 모멘트 (단위: Nm)

3.8 Camera 센서

카메라는 센서가 부착된 부분에서 바라보는 영상을 얻어올 수 있다.

```
void SetCamera(const char *name, const char *base_name, int width, int height, int fov);
CCamOgl *GetCamera (const char *name);
```

공통 인자:

- const char *name : 카메라로 사용할 물체의 이름 지정. 물체는 이미 생성되어 있어야 한다.

SetCamera

카메라를 설정한다.

- const char *base_name : 카메라를 고정할 물체의 이름 지정
- int width, height : 카메라의 영상 크기 (단위: pixel)
- int fov : 카메라의 화각 크기 (단위: deg)

GetCamera

카메라 이름으로부터 카메라 객체를 얻어온다.
만일 카메라 객체를 얻어왔다면, 카메라 객체로부터 이미지를 얻어오는 함수는 다음과 같다.

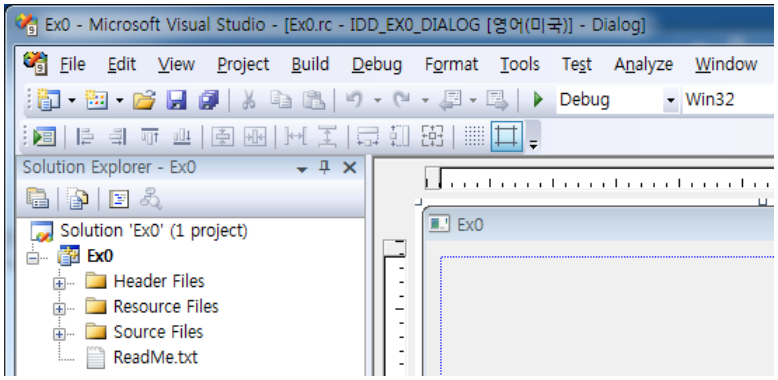
```
CCamOgl *cam = ode->GetCamera ("CAMERA");

const BYTE *img = cam->GetImage();
BITMAPINFO *bi = cam->GetBitmapInfo();
```

4 예제 프로젝트

4.1 프로젝트 생성

ODE 예제 프로젝트를 만든다. 예제 프로젝트를 만들 때 Project types로 MFC를 선택하고 Templates로 MFC Application을 선택한다. 그리고 Create directory for solution을 체크하여 솔루션과 프로젝트 폴더가 분리되도록 한다.

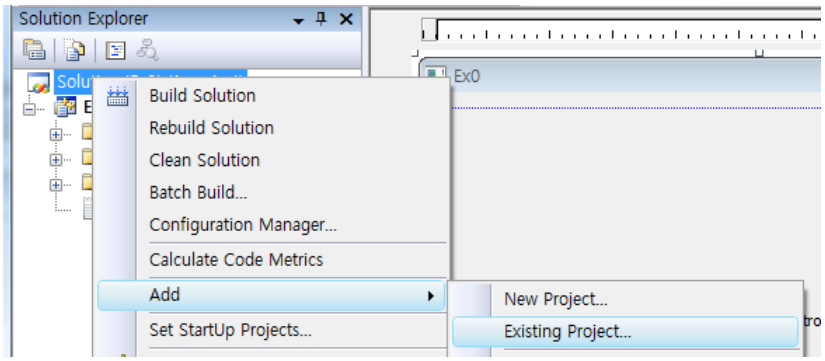


여기서는 C:₩temp 폴더에 Ex0라는 대화상자 기반 솔루션으로 만들었다.
그리고 솔루션을 만든 폴더에 ODE 라이브러리를 복사한다.

- 솔루션 이름: Ex0
- 솔루션 경로: C:₩temp₩Ex0
- 프로젝트 경로: C:₩temp₩Ex0₩Ex0
- ODE Library: C:₩temp₩Ex0₩ode-0.11.1

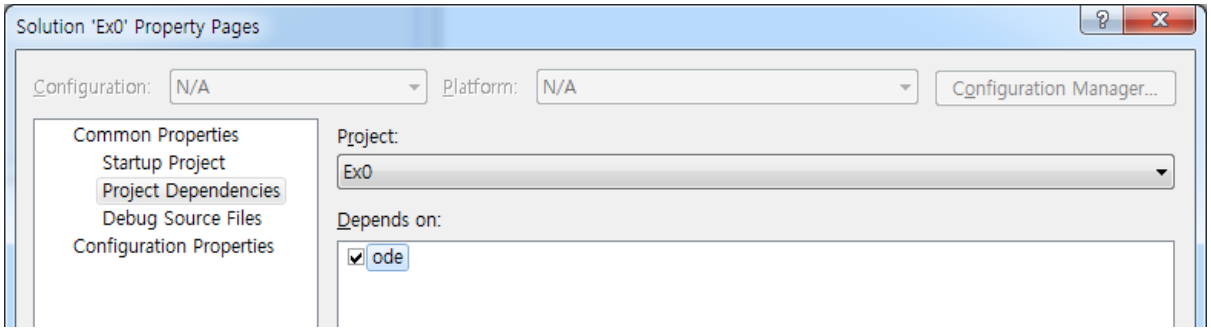
4.2 Solution 세팅

예제 프로젝트 솔루션에 ODE 라이브러리를 다음 그림과 같이 Solution -> Add-> Existing Project를 선택하여 ODE 라이브러리(ode.vcproj)를 현재의 solution에 추가한다.



- ODE Library경로: C:₩temp₩Ex0₩ode-0.11.1₩build₩vs2008₩ode.vcproj

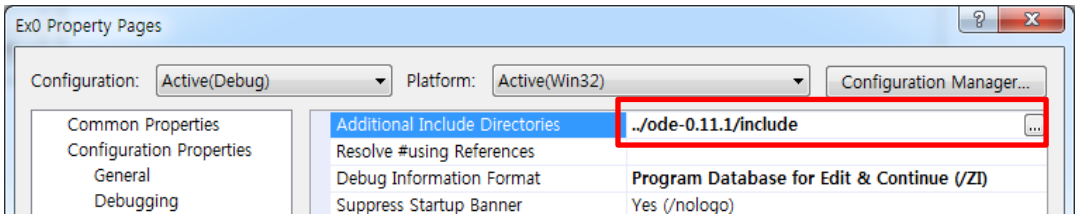
다음 그림과 같이 Solution의 Property pages에서 Project Dependencies를 선택한다. 여기서 Ex0 Project를 선택하고 ode를 체크 한다.



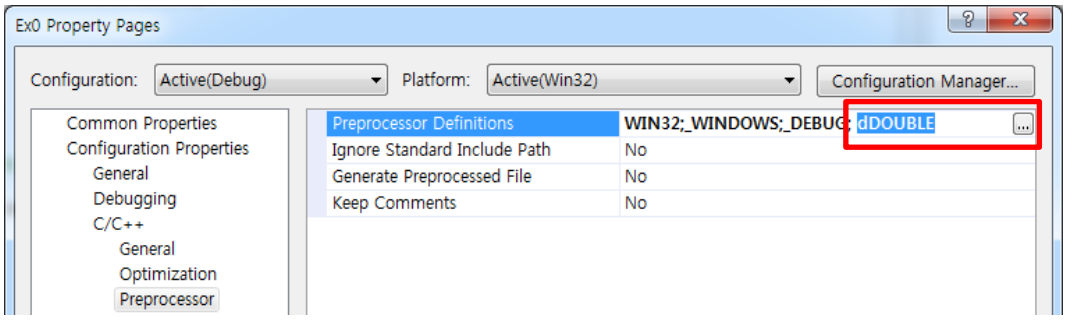
4.3 Project 세팅

VC++ 프로젝트에 Property Page에서 다음과 같이 설정한다.

다음 그림과 같이 C/C++ -> General -> Additional include Directories에 ../ode-0.11.1/include를 추가한다.



다음 그림과 같이 C/C++ -> Preprocessor -> Preprocessor Definitions에 dDOUBLE을 추가한다.

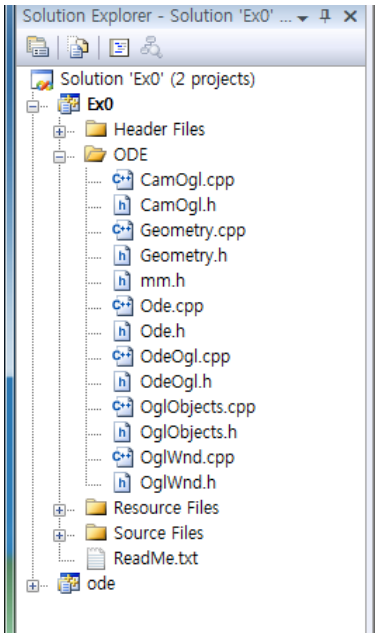


4.4 ODE Wrapping Source 추가

다음의 파일들을 Source Files에 추가 한다.

- CamOgl.cpp, CamOgl.h : 카메라를 생성하는 클래스
- Geometry.cpp, Geometry.h : 물체들의 형상을 가지는 구조체들
- mm.h: 간단한 수학 상수와 연산자들 정의
- Ode.cpp, Ode.h : ODE를 로봇에서 쉽게 이용할 수 있도록 레핑한 클래스
- OdeOgl.cpp, OdeOgl.h : ODE 시뮬레이션 결과를 OpenGL로 표시하는 클래스

- OglObjects.cpp, OglObjects.h : OpenGL에서 구, 실린더, 캡슐, 박스와 같은 형상을 그리기 위한 함수들
- OglWnd.cpp, OglWnd.h : OpenGL을 사용하여 3D randering을 담당하는 클래스



4.5 **코드 편집**

생성한 프로젝트의 Ex0Dlg.h에 Ode.h 파일과 OdeOgl.h 파일을 인클루드 한다.

```
#include "Ode.h"
#include "OdeOgl.h"
```

그리고 CEx0Dlg 클래스 내에서 COde와 COdeOgl 클래스의 객체 포인터를 선언한다.

```
// 물리엔진 ODE 를 사용하기 위한 클래스의 포인터
COde *_ode;
// 시뮬레이션 결과를 OpenGL 로 표시하기 위한 클래스의 포인터
COdeOgl *_ogl;
```

또, OnSize, OnDestroy, OnTimer 등 이벤트 핸들링 함수를 생성한다.

이제 Ex0Dlg.cpp 파일의 OnInitDialog () 함수에 코드를 추가한다.

```
    // OpenGL 로 그릴 화면의 크기를 가져온다.
    // 여기서는 클라이언트 영역 전체를 가져온다.
    CRect rect;
    GetClientRect(rect);

    // ODE 객체를 생성한다.
    _ode = new COde ();

    // 무게 2kg, 가로 0.2m, 세로 0.2m, 높이 0.2m 상자를 2 미터 높이에서 생성함, 그리고 구, 실린더, 캡슐 순으로...
    _ode->AddBox     ("box", 2., RGB(255, 0, 0),   0., 0., 2. , _DEG2RAD*45., _DEG2RAD*45., _DEG2RAD*0., 0.4, 0.4, 0.4);
    _ode->AddSphere  ("sph", 2., RGB(0, 255, 0),   0., 0., 2.5, _DEG2RAD*45., _DEG2RAD*45., _DEG2RAD*0., 0.2);
    _ode->AddCylinder("cyl", 2., RGB(0, 0, 255),   0., 0., 3. , _DEG2RAD*45., _DEG2RAD*45., _DEG2RAD*0., 0.2, 0.4);
    _ode->AddCapsule ("cap", 2., RGB(255, 255, 0), 0., 0., 3.5, _DEG2RAD*45., _DEG2RAD*45., _DEG2RAD*0., 0.2, 0.2);

    // ODE 를 그리기 위한 OpenGL 객체를 생성한다.
    _ogl = new COdeOgl(_ode, "ODE", rect, this, 90000);
    _ogl->_showAxis = true;

    SetTimer (1000, 10, NULL);
```

OnSize() 함수와 OnDestroy() 함수, OnTimer() 함수에 다음과 같이 코드를 추가한다.

```
void CEx0Dlg::OnSize(UINT nType, int cx, int cy)
{
    CDialog::OnSize(nType, cx, cy);

    // 메인 윈도우의 크기가 바뀌는 경우, OpenGL 윈도우의 크기를 조절한다.
    _ogl->MoveWindow (0, 0, cx, cy);
}

void CEx0Dlg::OnDestroy()
{
    CDialog::OnDestroy();

    _ogl->DestroyWindow ();
    delete _ogl;

    delete _ode;
}


void CEx0Dlg::OnTimer(UINT_PTR nIDEvent)
{
    // ODE 시뮬레이션을 한 스탭 진행한다.
    _ode->Step (0.01);
    // 시뮬레이션 결과를 OpenGL 윈도우에 업데이트 한다.
    _ogl->UpdateWnd ();

    CDialog::OnTimer(nIDEvent);
}
```