Geometry Dispatching

Kenny Erleben

Department of Computer Science

University of Copenhagen

Objects

```
class Object
{
public:
  geometry_type * m_geometry;
};

std::vector<Object> scene;
```

Broad-Phase

```
void broad_phase(
  std::vector<Object> & scen
, std::vector<pair<geometry_type*,geometry_type*> > & geometry_pairs
                )
{
  geometry_pairs.clear();
  for(unsigned i=0;i< scene.size();++i)
    for(unsigned j=i+1;j< scene.size();++j)
      geometry_pairs.push_back(
         make_pair(scene[i].m_geometry, scene[j].geometry)
      );
}
```

Collision Detection

```
void collision_detection(std::vector<Object> & scene)
{
  typedef std::vector<pair<geometry_type*,geometry_type*> > pair_conta
  pair_container pairs;

  broad_phase(scene,pairs);

  pair_container::iterator p   = pairs.begin();
  pair_container::iterator end = pairs.end();

  for(;p!=end;++p)
    narrow_phase(p->first,p->second);
}
```

The Geometry Type

```
class Geometry
{
public:
   virtual std::string get_type() const = 0;
   //--- Bunch of other things here...
};

class Sphere : public Geometry
{
public:
  std::string get_type() const {  return ‘‘sphere’’; }
};

class Box : public Geometry
{
public:
  std::string get_type() const {  return ‘‘box’’; }
};
```

Collision Handlers

```
void test_box_sphere(Box const & A, Sphere const & B)
{
  std::cout << ‘‘box sphere test’’ << std::endl;
}

void test_box_box(Box const & A, Box const & B)
{
  std::cout << ‘‘box box test’’ << std::endl;
}

void test_sphere_sphere(Sphere const & A, Sphere const & B)
{
  std::cout << ‘‘sphere sphere test’’ << std::endl;
}
```

Narrow-Phase

```
void narrow_phase(geometry_type * A,geometry_type * B)
{
  if( A->get_type()==''box'' && B->get_type() == ''sphere'')  {
     Box * box    = static_cast<Box *>(A);
     Sphere * sphere = static_cast<Sphere *>(B);
     test_box_sphere(*box,*sphere);
  }  else if( A->get_type()==''box'' && B->get_type() == ''box')  {
     Box * boxA = static_cast<Box *>(A);
     Box * boxB = static_cast<Box *>(B);
     test_box_box(*boxA,*boxB);
  } else if( A->get_type()==''sphere'' && B->get_type() == ''sphere')
     Sphere * boxA = static_cast<Sphere *>(A);
     Sphere * boxB = static_cast<Sphere *>(B);
     test_sphere_sphere(*sphereA,*sphereB);
  }  else  {
     std::cerr << ''sorry could not decode geometry types'' << std::er
  }
}
```

Another Hacky Approach

```
class Object
{
public:
  typedef enum {sphere, box, ...} geometry_types;
  geometry_type m_type;
  void * m_geometry;
};
```

The Task

- This is a unsolved problem, there is no known exisiting best known design pattern for this!
- Try to come up with a different design approach
- Try to attack performance, maintance, and modularity problems