11/19/2014

Double Dispatch
(New Scene API in VTK 6.x)

Sandy Engelhardt

dkfz.
DEUTSCHES
KREBSFORSCHUNGSZENTRUM
IN DER HELMHOLTZ-GEMEINSCHAFT

Double dispatch vs. Single dispatch

Double dispatch

- is a mechanism that dispatches a function call to different concrete functions **depending on the runtime types of two objects involved in the call**
- is useful in situations where the choice of computation depends on the runtime types of its arguments

Single dispatch (= virtual function calls)

- the concrete function that is called from a function call in the code **depends on the dynamic type of a single object**

Example Situations

- ***Sorting a mixed set of objects***
  Deciding if one element comes before another element requires knowledge of both types and possibly some subset of the fields.

- ***Event handling*** systems that use both the *event type* and the *type of the receptor* object in order to call the correct event handling routine.

- ***Adaptive collision algorithms*** in a game environment usually require that collisions between different objects be handled in different ways: collision between a spaceship and an asteroid is computed differently from the collision between a spaceship and a spacestation.

Let‘s fly to the moon …

hit

```
class SpaceShip {};
```

```
class Asteroid {} ;
```

```
class ApolloSpacecraft :
public SpaceShip {};
```

```
class ExplodingAsteroid :
public Asteroid {};
```

Let‘s fly to the moon …

```
class Asteroid
{
public:
  virtual void CollideWith(SpaceShip&)
  {
    cout << "Asteroid hit a SpaceShip" << endl;
  }
  virtual void CollideWith(ApolloSpacecraft&)
  {
    cout << "Asteroid hit an ApolloSpacecraft" << endl;
  }
};


class ExplodingAsteroid : public Asteroid
{
public:
  virtual void CollideWith(SpaceShip&)
  {
    cout << "ExplodingAsteroid hit a SpaceShip" << endl;
  }
  virtual void CollideWith(ApolloSpacecraft&)
  {
    cout << "ExplodingAsteroid hit an ApolloSpacecraft" << endl;
  }
};
```

Let‘s fly to the moon …

Suppose `SpaceShip` and `ApolloSpacecraft` both have the function

```
virtual void CollideWith(Asteroid& inAsteroid)
{
 inAsteroid.CollideWith(*this);
}
```

If you have

```
Asteroid theAsteroid;
SpaceShip theSpaceShip;
ApolloSpacecraft theApolloSpacecraft;
ExplodingAsteroid theExplodingAsteroid;

SpaceShip& theSpaceShipReference = theApolloSpacecraft;
Asteroid& theAsteroidReference = theExplodingAsteroid;
```

theSpaceShipReference.CollideWith(theAsteroid);

```
A) "Asteroid hit a SpaceShip"
B) "Asteroid hit an ApolloSpacecraft"
C) "ExplodingAsteroid hit a SpaceShip"
D) "ExplodingAsteroid hit an ApolloSpacecraft"
```

B) “Asteroid hit an ApolloSpacecraft”

theSpaceShipReference.CollideWith(theAsteroidReference);

D) “ExplodingAsteroid hit an ApolloSpacecraft”

New Scene API in VTK

- double dispatch to enable runtime extension
- basic rendering API has two base classes - `Node` and `Visitor`
- `Visitor` moves through a graph of `Nodes`

```
class Node
{
public:
  Node();
  virtual ~Node();

  /** Accept a visit from our friendly visitor. */
  virtual void accept(Visitor &) { return; }

  /** Traverse any children the node might have, and call accept on them. */
  virtual void traverse(Visitor &) { return; }

  /** Ascend to the parent and call accept on that. */
  virtual void ascend(Visitor &);

  /** Get a pointer to the node's parent. */
  const GroupNode * parent() const;
  GroupNode * parent();
};
```

Visitor pattern

```
class Visitor
{
public:
  Visitor();
  virtual ~Visitor();

  /** The overloaded visit functions, the base versions of which do nothing. */
  virtual void visit(Node &) { return; }

  virtual void visit(GroupNode &) { return; }

  virtual void visit(GeometryNode &) { return; }

  virtual void visit(TransformNode &) { return; }

  virtual void visit(Drawable &) { return; }

  virtual void visit(MeshGeometry &) { return; }
};
```

Render Passes

```
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
applyProjection();

RenderVisitor visitor(m_camera);
// Setup for opaque geometry
visitor.setRenderPass(OpaquePass);
glEnable(GL_DEPTH_TEST);
glDisable(GL_BLEND);
m_scene.rootNode().accept(visitor);

// Setup for transparent geometry
visitor.setRenderPass(TranslucentPass);
glEnable(GL_BLEND);
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
m_scene.rootNode().accept(visitor);

// Setup for 3d overlay rendering
visitor.setRenderPass(Overlay3DPass);
glClear(GL_DEPTH_BUFFER_BIT);
m_scene.rootNode().accept(visitor);

// Setup for 2d overlay rendering
visitor.setRenderPass(Overlay2DPass);
visitor.setCamera(m_overlayCamera);
glDisable(GL_DEPTH_TEST);
m_scene.rootNode().accept(visitor);
```

Once inside the accept method, it will typically call visit as in the GroupNode

```
void GroupNode::accept(Visitor &visitor)
{
  visitor.visit(*this);
}
```

This will then call the GroupNode version of the visit member on the visitor (in this case the RenderVisitor),

```
void RenderVisitor::visit(GroupNode &group)
{
  group.traverse(*this);
}
```

This causes the node to call accept on all child nodes,

```
void GroupNode::traverse(Visitor &visitor)
{
  for (std::vector<Node *>::iterator it = m_children.begin();
       it != m_children.end(); ++it)
  {
    (*it)->accept(visitor);
  }
}
```

The next interesting action is to call the visit for a Drawable,

```
void RenderVisitor::visit(Drawable &geometry)
{
  if (geometry.renderPass() == m_renderPass)
  {
    geometry.render(m_camera);
  }
}
```

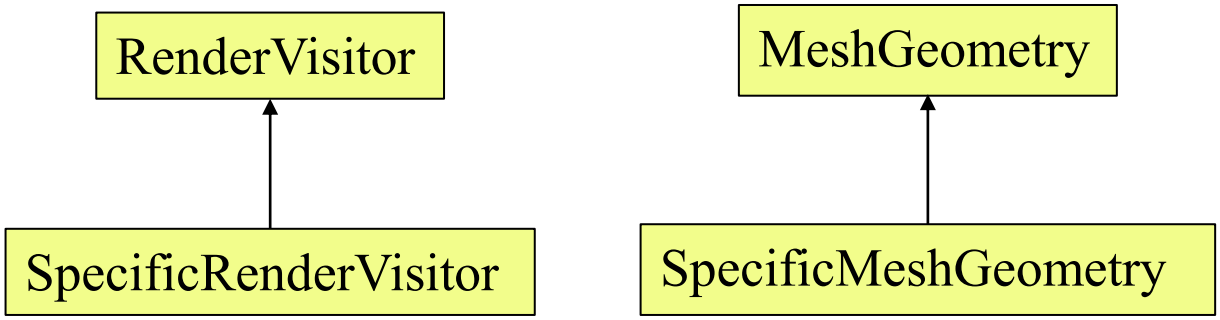
Similarly, a possible implementation for a transform node is,

```
void RenderVisitor::visit(TransformNode &transform)
{
  Camera old = m_camera;
  m_camera.setModelView(m_camera.modelView() * transform.transform());
  transform.traverse(*this);
  m_camera = old;
}
```

What‘s new?

- Double dispatch enables users of the API to **register new node types or visitor types at runtime**, and traversal would always execute the **most derived** form of a type in the hierarchy

- Example: this allows us to **override the RenderVisitor** for a derived SpecificMeshGeometry object, but leave the default implementation of RenderVisitor call for the less derived MeshGeometry types



Thank you for your attention!