ISBN 978-89-6211-453-9 98560

GLOVE Drawing Context 관리

(GLOVE Drawing Context Management)

김 민 아 (petimina@kisti.re.kr)

한 국 과 학 기 술 정 보 연 구 원

Korea Institute of Science & Technology Information

목차

1. 개 요 ····· 1

2. OSG (Open Sene Graph) ····· 2

가. Scene Graph의 개념 ····· 2

나. OSG Classes ····· 3

3. Drawing Context ····· 5

가. Drawing Object의 관리 ····· 6

1) Drawing Object ····· 6

2) Transform Matrix ····· 6

3) Drawing Object의 관계 ····· 6

4) 선택된 Drawing Object의 리스트 ····· 6

나. User Operation의 History관리 ····· 6

1) Event History ····· 6

2) Command History ····· 7

4. Drawing Object Tree(dotree) ····· 7

가. Drawing Object Tree의 개념 ····· 7

나. Drawing Object Tree의 Class들 ····· 9

1) Drawing Object Tree의 class inheritance diagram ····· 9

2) dotree::Node ····· 10

3) dotree::Group ····· 10

4) dotree::Transform ····· 10

5) dotree::Object ····· 10

다. Drawing Object Tree의 사용 방법 ····· 11
1) root node 생성 ····· 11
2) transform node 생성 ····· 11
3) object node 생성 ····· 11
4) Add child node (자식 노드 추가) ····· 11
5) Remove child node (자식 노드만 삭제) ····· 12
6) Remove children nodes ····· 12
7) Transform child node ····· 12
8) 자식 노드 개수 ····· 13
9) 자식 노드와 관련된 기능들 ····· 13
10) 탐색 (이름으로 자손노드까지 탐색) ····· 13
11) 노드 이름 출력 ····· 13
12) vtkActor list 가져오기 ····· 14

5. 응용에 의존적인 Context 관리 ····· 14
1) Command and Event History ····· 15
2) Application Object Context ····· 15

6. 결론 ····· 15
7. 참고문헌 ····· 16

그림 차례

[그림 1-1] A scene graph,consisting of a road and a trunk ····· 2
[그림 1-2] A scene graph, consisting of a road and a translated truck ····· 2
[그림 1-3] A scene graph, consisting of a road, a truck, and a pair of boxes ····· 3
[그림 1-4] Instances of OSG classed derived osg::Node ····· 4
[그림 1-5] An OSG scene graph, consisting of a road and a translated truck ····· 4
[그림 1-6] An OSG graph, conssting of a road, a trunk, and a pair of boxes ····· 5
[그림 2-1] A drawing object tree in GLVOE ····· 8
[그림 2-2] A drawing Object tree after deleting a fuselage ····· 8
[그림 2-3] A drawing object tree, consisting of a LoadButton and a Blade, and a Plane on the Blade ····· 9
[그림 2-4] Drawing object tree class inheritance ····· 10

1. 개 요

어떤 대상을 가시화 하여 화면에 그리는 컴퓨터 그래픽스 응용에서, 한 순간에 화면에 그려진 대상들은 하나의 scene을 구성한다. 한 scene에는 그려진 오브젝트은 초기의 위치가 있다. 초기의 위치는 사용자의 조작에 따라 그 위치를 옮길 수 있다. 또한 scene에는 오브젝트가 추가 될 수 있고, 삭제 될 수 있으며, 오브젝트를 선택하여 변형할 수도 있다.

응용 프로그램이 이런 scene의 변화를 해석하고, 사용자의 조작에 반응하기 위해서는 그려진 대상인 이들 오브젝트의 리스트를 유지하고 관리해야 한다.

이러한 scene을 관리하기 위해 컴퓨터 그래픽스 응용들은 여러 가지 데이터 구조를 사용해 왔다. OSG(Open Scene Graph)는 그 대표적인 예로 하나의 scene 위에 그려진 오브젝트를 Graph의 형태로 관리한다.

그러나, 응용이 scene에서 관리해야 할 것은 이뿐만이 아니다. 사용자가 어떤 조작을 했었는지 어떤 변화가 있었는지 사용자의 인터렉션에 관한 정보도 유지해야만, 사용자 메뉴와 연관해 scene을 제대로 관리할 수 있다. 본 문서에서는 이러한 모든 정보를 Drawing Context라 정의한다.

GLOVE (GLObal Virtual Environment for collaborative research)는 KISTI에서 개발 중인 대용량 데이터를 효율적으로 가시화하여 공유하는 협업 가시화 환경이다. GLOVE의 대상 데이터는 계산과학의 시뮬레이션 결과이거나, 과학 실험의 결과이다. 이들 데이터들은 응용에 따라 여러 가지 가시화 기술과 방법으로 화면에 그려진다. 또한, GLOVE 는 VRJuggler와 VTK라는 가시화 개발 툴로 오브젝트를 화면에 그린다. 이 때문에, OpenGL context에 의존적인 OSG를 사용하는 데에 어려움이 있다. 따라서, VRJuggler와 VTK의 프레임워크 내에서 context를 관리하는 또 다른 데이터 구조가 필요하다.

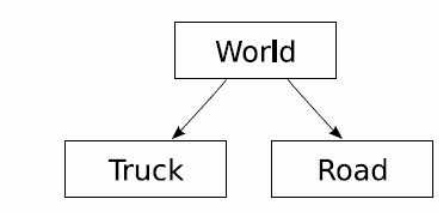
본 문서에서는 GLOVE의 특수성을 고려하여, drawing context를 관리하는 새로운 데이터 구조를 설계하고, 구현한다. 먼저 scene을 잘 정의한 OSG의 구조를 살펴봄으로써, scene을 구성하는 요소와 고려해야 할 점들에 대해 알아본다. 다음으로, GLOVE 개발과정에서 필요한 Drawing Context를 정의하고, 이를 관리하기 위한 데이터 구조를 설계한다. 마지막으로 구현된 데이터 구조의 사용 방법에 대해 기술한다.

2. OSG (Open Sene Graph)

본 절에서는 Open Scene Graph의 개략적 개념에 대해 알아봄으로써, drawing context 중 하나의 scene을 오브젝트로 표현하는 방법에 대해 살펴본다.

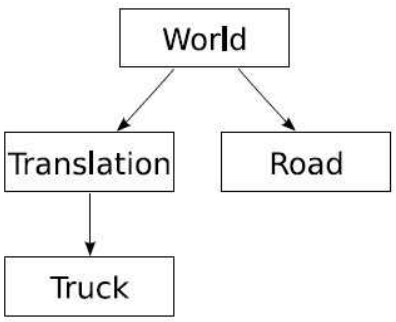
가. Scene Graph의 개념

Scene Graph는 이름이 의미하는 바와 같이 scene을 관리하는 하나의 directed acyclic graph 이다. 만약 하나의 scene이 길과 트럭으로 구성되어 렌더링 된다고 한다면, 이런 장면을 표현하는 scene graph는 [그림 1-1]과 같이 그려볼 수 있다.



[그림 1-1 A scene graph,
consisting of a road and a trunk]

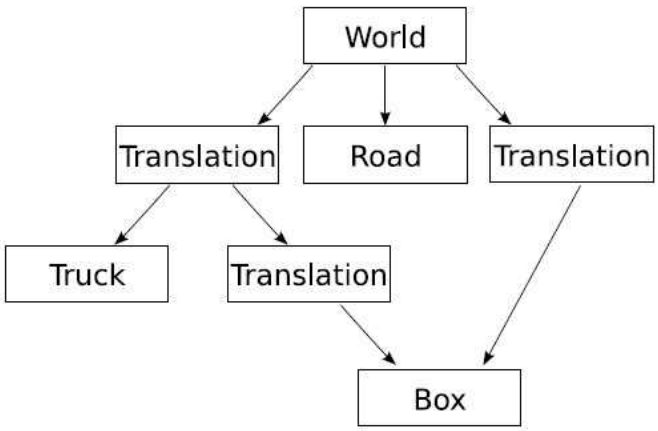
그러나, 이 그래프는 geometry를 표현하지 못한다. 이 경우, [그림 1-2]와 같이 translation을 표현하는 하나의 노드를 더할 수 있다.



[그림 1-2 A scene graph, consisting of
a road and a translated truck]

그런데 이렇게만 한다면 scne은 그래프가 아니라 트리만으로 표현이 가능하다.

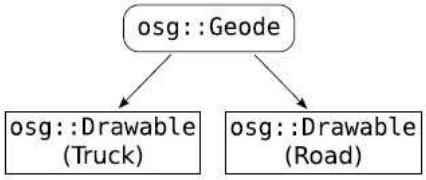
그렇다면 왜 Open Scene Graph 인가? 다음 경우를 생각해 보자. 이 scene에 트럭과 임의의 위치에 각각 하나의 박스를 더한다고 가정해보자. 그 두 개의 박스는 정확하게 동일하며 단지 위치만 하나는 트럭 위 하나는 임의의 위치에 있을 뿐이다. 그렇다면 scene 그래프에서는 하나의 노드만 만들고, 서로 다른 translation이 그 박스를 두 번 참조하는 트릭을 쓰면 된다. [그림 1-3]은 트릭을 사용한 scene graph를 보여준다. 이 그림은 이제 더 이상 트리가 아닌 그래프이다.



[그림 1-3 A scene graph, consisting of a road, a truck, and a pair of boxes]

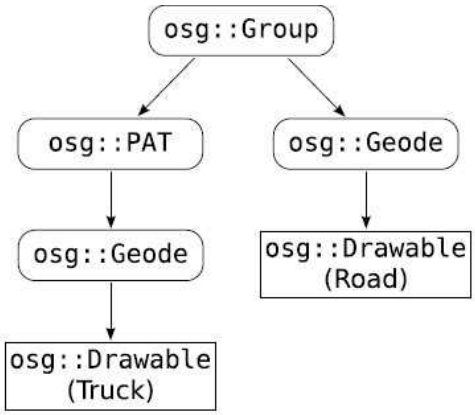
나. OSG Classes

이상에서는 일반적인 scene graph에 대해서 살펴보았다. 이를 구현하기 위해 OSG는 클래스를 만들고 이를 통해 scene graph를 표현한다. OSG는 이를 osg::Node 클래스로 이를 구현한다. 정확히 말하면, osg:Node의 subclass를 통해서다. 이 세 개의 클래스는 osg::Group, osg::PositionAttributeTransform, osg::Drawable이다. 렌더링할 수 있는 어떤 오브젝트를 osg::Drawable의 인스턴스로 표현할 수 있다. 그러나, 좀 더 정확히 말하면, osg::Drawable은 노드가 아니다. 따라서, OSG에 바로 붙일 수 없다. osg::Drawable을 붙이려면 “gemometry node" 인 osg::Geode가 필요하다. [그림 1-3]은 [그림 1-1]을 OSG 클래스로 표현한 그림이다.



[그림 1-4 Instances of OSG classed derived osg::Node]

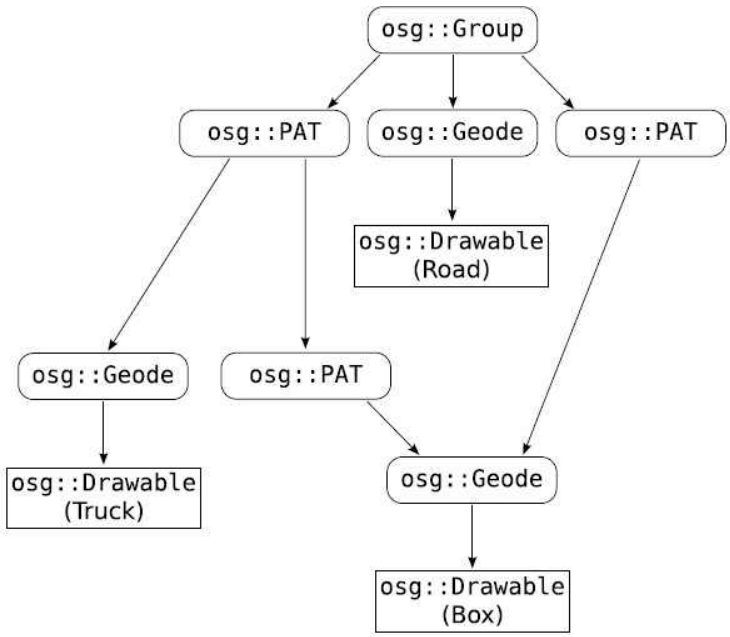
그림에서 보듯이 하나의 osg::Geode 에 여러개의 osg::Drawable이 붙여 질 수 있다. 그러나, [그림 1-1]에서와 마찬 가지로 [그림 1-4]의 트럭은 잘못된 위치에 놓일 수 있다. 해결책은 트럭에 translation을 주는 것이다.



[그림 1-5 An OSG scene graph, consisting of a road and a translated truck]

[그림 1-5]는 translation을 반영한 트럭과 길로 구성된 OSG이다. osg::PAT는 osg::PositionAttributeTransform를 의미한다. osg::Geode와 osg::Group은 geometry를 가진 하나의 scene graph를 구성할 수 있다. osg::PAT는 위치 외에 osg::Drawable의 인스턴스에 자세나 크기 등과 같은 속성을 부여할 수 있다. 바꿔 말하면, OpenGL의 glRotate(), glScale(), glTranslate()이 수행하는 기능과 같은 역할을 한다. 따라서, osg::Drawable을 움직이기 위해서는 osg::Geode만 사용하기 보다는 osg::PAT를 사용하는 것이 좀 더 일반적이다. 그러나, osg::Geode는 그 자식으로 osg::PAT를 가질 수 없다. 이 때문에 최상단의 노드는 osg::PAT와 osg::Geode를 모두 자식으로 둘 수 있는 osg::Group이 위치한다. [그림 1-6]은 [그림 1-3]의 scene graph를 OSG이 클래스로 표현한 것이다.

서로 다른 위치에 있는 Box를 다른 osg::PAT를 반영하여 scene graph로 표현하고 있다.



[그림 1-6 An OSG graph, conssting of a road, a trunk, and a pair of boxes]

OSG는 OSG의 geometry에 관련된 속성들을 다루는 StateSet이라는 매우 강력한 클래스를 지원한다. 그러나, 잊지 말아야 할 사실은 OSG가 매우 강력한 라이브러리이긴 하지만, 결국 OpenGL의 wrapper 라는 사실이다. 즉, OpenGL을 이미 한번 wrapping 하여 자신의 상태를 스스로 관리하고 있는 VTK를 사용할 경우 이를 반영할 수 없다. VTK를 위해 구현된 OSG는 개발 중이긴 하나 아직은 존재하지 않는다.

3. Drawing Context

GLOVE는 기본적인 가시화 협업 프레임워크 위에 응용 맞춤형 사용자 인터페이스를 구현한다. 이것은 컴퓨터 그래픽스 측면에서 화면에 가시화되는 오브젝트와 그 가시화 방법이 다양하고, scene을 구성하는 형태나 scene에 가해지는 사용자의 조작 요구사항이 달라질 수 있다는 것을 의미한다. 따라서, 하나의 scene에 가해 질 수 있는 모든 사용자 인터렉션과 오브젝트의 관계를 추상화하여 관리할

수 있는 Drawing Context가 필요하다. 본 절에서는 GLOVE의 이러한 특징을 고려한 Drawing Context의 구성요소에 대해 기술한다.

가. Drawing Object의 관리

1) Drawing Object

GLOVE와 같이 VTK로 가시화를 구현하는 응용의 경우 그 최종 산물은 vtkActor이다. vtkActor를 vtkRenderer에 더해 주면 화면에 오브젝트가 그려지는 것이다. 오브젝트를 그리기 위한 속성들은 vtkActor와 vtkActor를 위한 vtkMapper에서 이미 모두 설정이 끝난 상태이다. 따라서, GLOVE에서 최종적으로 관리해야 할 것, 즉 OSG의 leaf node인 osg::Drawable에 해당하는 것은 vtkActor이다.

2) Transform Matrix

vtkActor는 scale, rotate, translation을 위해 자신만의 transform matrix를 가진다. 따라서, drawing context는 drawing object와 transform matrix의 쌍을 함께 관리하여야 한다.

3) Drawing Object의 관계

때로 오브젝트들은 상호관계를 맺고 있을 경우가 있다. 사람의 뼈와 혈과 데이터는 서로 다른 vtkActor로 표현되지만, 이동을 하거나 회전을 할 때 함께 움직여야 한다. Drawing context는 하나의 scene에 이러한 관계를 묘사할 수 있어야 한다.

4) 선택된 Drawing Object의 리스트

사용자는 하나의 scene을 구성하고 있는 오브젝트를 선택하여 삭제 하거나, 변경하거나, 혹은 애니메이션을 요청할 수도 있다. 이러한 선택이 일어날 경우 선택된 오브젝트는 유일한 오브젝트 id 로 관리되어야 하며, 여러 개를 선택할 경우 그 리스트를 유지할 수 있어야 한다.

나. User Operation의 History관리

1) Event History

GLOVE는 메뉴를 VRJuggler의 프레임워크위에 VTK를 사용하여 구현된다. 이 때문에 사용자 입력은 VTK의 event로 처리된다. VR환경의 입력도구의 상태 변

화 (wand의 움직임, obejct의 선택 등) 또한 event를 발생시켜 처리한다. 이러한 이벤트의 history도 현재의 scene을 구성하는 하나의 context이다.

2) Command History

그래픽스 도구를 사용하는 사용자는 어떤 명령을 수행하고 난 뒤, 자신이 방금 사용한 명령을 취소하고 싶기도 하고, 이전의 어느 상태까지 돌려놓기를 원하기도 한다. 현재 scene을 만들기 위해 수행한 명령의 순서를 기억하고 있다면, 이러한 "되돌리기"를 수행할 수 있다. 이러한 명령은 응용에 따라 달라 질 수 있으나, 가시화를 위해 수행하는 명령은 어느 정도 일반화가 가능하므로, drawing object와 명령의 조합으로 command history를 만듦으로써, 일반화된 command history를 관리할 수 있다.

4. Drawing Object Tree(dotree)

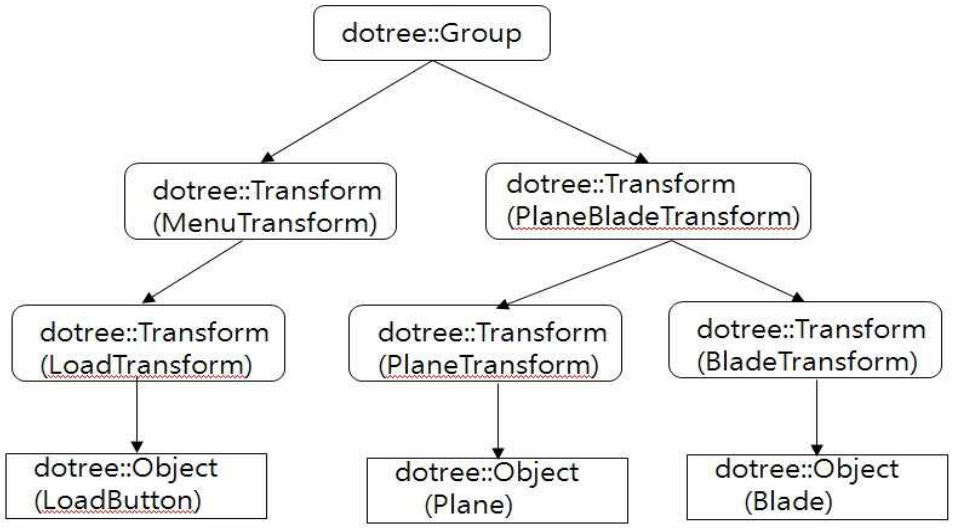
GLOVE는 응용의 특성상 정확하게 [그림 2-5]의 scene graph가 보여 주는 동일한 모양을 가진 오브젝트가 다른 위치에 존재하는 일은 없다. 따라서, GLOVE의 오브젝트들을 관리하는데 graph 구조보다는 좀 더 간단한 tree 구조를 선택한다. Drawing Object Tree는 3절에서 언급한 요소들 중 Drawing Object를 관리하기 위한 tree 형태의 자료 구조이다.

하나의 scene에서 오브젝트를 지울 경우, 이 오브젝트는 Drawing Object Tree에서도 지워져야 한다. 또한 여러 개의 vtkActor로 구성된 의미론적으로 하나인 오브젝트들도 사용자의 요구에 따라 그룹으로 혹은 하나로 지워질 수 있어야 한다.

가. Drawing Object Tree의 개념

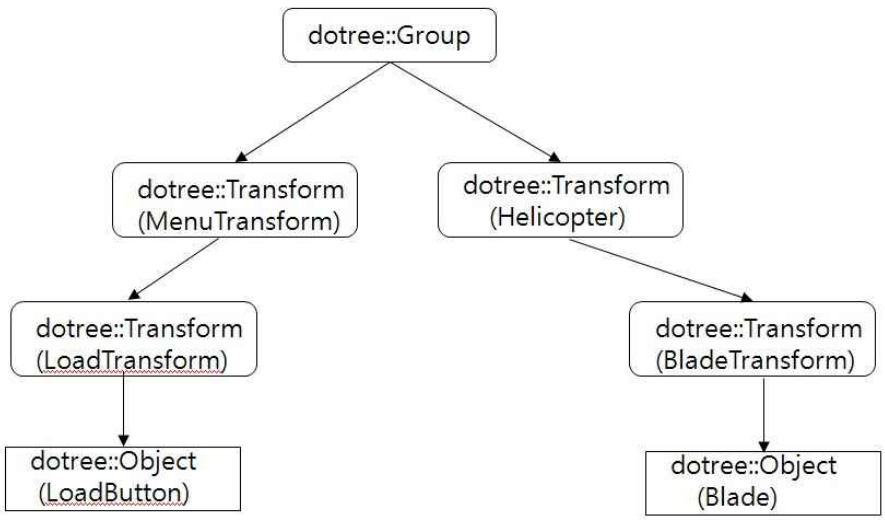
Drawing Object Tree는 현재 scene 에 존재하는 모든 오브젝트를 관리한다. GLOVE의 경우 화면의 한 scene 에는 응용 데이터를 가시화한 오브젝트만 존재하는 것이 아니라 메뉴와 사용자 인터페이스를 위한 대화박스들도 존재 한다. 이들은 사용자의 조작에 따라 나타났다 사라졌다를 반복하며, 이동이 가능한 오브젝트들이다. GLOVE에서는 이들 모두를 Drawing Object Tree내에서 관리한다.

이들 오브젝트들은 필요에 따라 하나의 그룹으로 분류되며 그룹단위로 지워지기도 하고 transform matrix를 적용하기도 해야 한다.



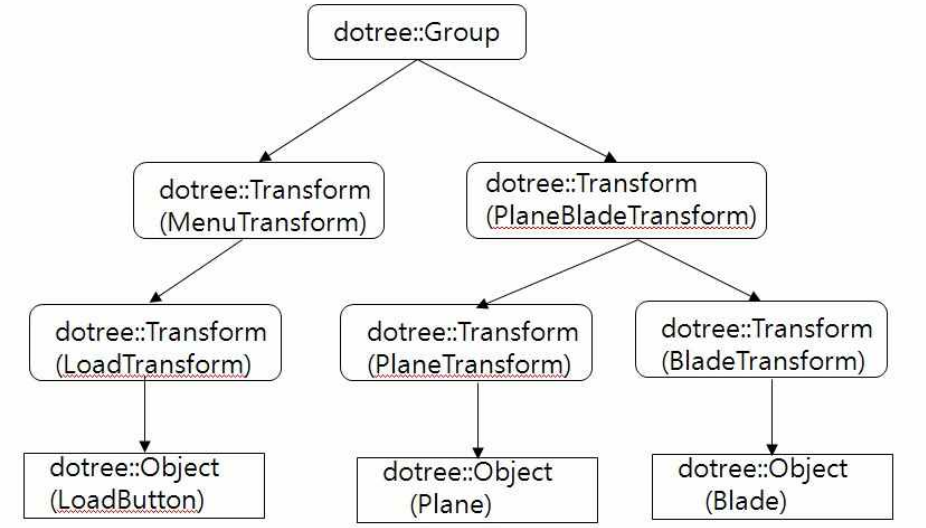
[그림 2-1 A drawing object tree in GLVOE]

[그림 2-1]은 로터 시뮬레이션 데이터를 위한 응용 맞춤형 사용자 인터페이스를 제공하는 GLOVE의 한 scene을 Drawing Object Tree로 표현한 것이다. 그림에서 보듯, 메뉴는 메뉴를 위한 tranfrom이 존재해야 하고, 블레이드와 동체는 각각의 transform이 존재하며 이를 하나로 묶는 transform이 그 위에 존재한다. 블레이드와 동체를 포함하는 헬기 전체를 이동할 경우 HelicopterTransform의 값을 변경하면 되고, 각각을 이동할 경우 바로 상위의 transform을 변경하면 된다. 메뉴는 메뉴 전체를 이동할 수 있는 transform이 따로 존재하고 그 아래에 메뉴 아이템들이 위치한다. 각 노드들은 FuselageTransform과 같은 고유의 이름으로 구분되며, 이 이름은 검색을 위해서 사용한다. dotree::Transform은 OSG의 osg::PAT와 유사하며, dotree::Group은 osg::Group과 유사하다.



[그림 2-2 A drawing Object tree after deleting a fuselage]

[그림 2-1]에서 사용자가 화면에 동체를 제거하고 블레이드만 보여주기를 원한다면, [그림 2-2]와 같이 Drawing Object Tree에서는 동체의 transform과 동체를 함께 삭제할 것이다. GLOVE는 가시화한 오브젝트의 내부 데이터를 보기 위해 하나의 plane을 생성해 오브젝트를 자르기도 한다. 이럴 경우, 그 plane은 DOT의 관점에서 하나의 오브젝트이며, 이 plane은 원하는 지점을 찾기 위해 움직일 수 있다. 또, 위치가 결정되면, 그 데이터를 좀 더 자세히 보기위해 단면을 확대할 수도 있고, plane과 오브젝트를 함께 움직일 수도 있다. [그림 2-3] 은 plane을 포함한 DOT를 보여준다.

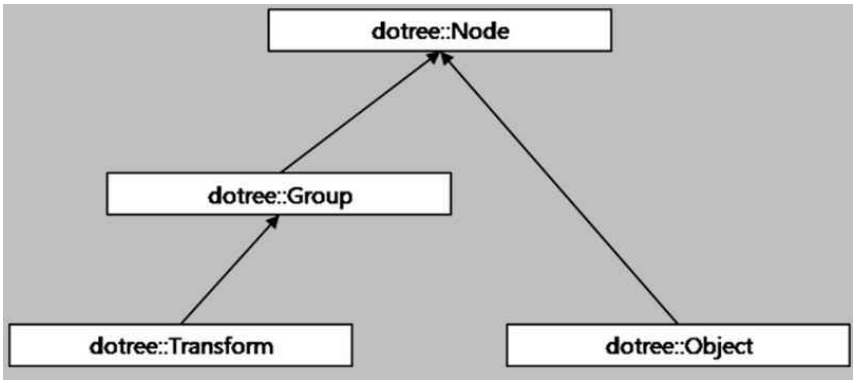


[그림 2-3 A drawing object tree, consisting of a LoadButton and a Blade, and a Plane on the Blade]

나. Drawing Object Tree의 Class들

1) Drawing Object Tree의 class inheritance diagram

Drawing Object Tree(이하 DOT)의 클래tm는 모두 4개로 [그림 2-4]는 DOT의 class inheritance diagram 이다. dotree::Node는 모든 node의 대표 속성을 가진 parent 클래스이고, dotree::Group은 자식노드를 가질 수 있다. dotree::Transform은 자식노드를 가질 수 있는 dotree::Group을 상속받는다.



[그림 2-4 Drawing object tree class inheritance]

2) dotree::Node

모든 클래스의 부모 클래스로 DOT내에서 하나의 노드를 유일하게 구분할 수 있는 구분자인 name과 모든 DOT의 노드들이 공통으로 가질 수 있는 parent를 속성으로 가진다. 또한, 이들 속성을 다룰 수 있는 operation을 가진다.

3) dotree::Group

DOT는 트리구조이므로 잎 노드가 아닌 노드들은 자식을 가질 수 있다. dotree::Group은 children을 속성으로 가지고 이들 children을 관리한다. 자식은 dotree::Transform일 수도 있고, dotree::Object일 수도 있고, 다른 dotree::Group일 수도 있다. 하나의 자식을 더할 때, name의 중복도 check해 더해지는 노드의 name이 DOT내에서 유일하도록 관리해 준다.

4) dotree::Transform

dotree::Group을 상속받아 자식노드로 dotree::Transform이나 dotree::Object를 가질 수 있다. 또한 속성으로 VTK의 transform matrix인 vtkTransform을 가진다. 이 matrix는 dotree::Transform의 모든 자식노드인 vtkActor에 반영될 것이다.

5) dotree::Object

dotree::Object는 잎노드로 현재는 vtkActor만을 속성으로 가진다. 그러나, GLOVE에서 관리되어야 할 Drawing Object는 vtkActor만 있는 것은 아니다. 이들 vtkActor들을 grouping하여 하나의 기능을 수행하는 오브젝트로 만든 vjWidget들도 존재한다. 따라서, 추후 vjWidget도 잎노드로 추가되어야 한다.

다. Drawing Object Tree의 사용 방법

1) root node 생성

```
dotree::Group* root = dotree::Group::New();
root->SetName("root");
```

트리형태로 오브젝트를 관리하기 위해서 먼저 root 노드를 만들어야 한다. dotree::Group만이 DOT의 root가 될 수 있다. dotree::Group을 생성한 다음 SetName으로 name을 지정한다.

2) transform node 생성

```
dotree::Transform* transform = dotree::Transform::New();
transform->SetName("trans_child1");
transform->SetTransform( vtkTransform );
```

dotree::Transform 노드는 생성과 함께 VTK의 transform matrix인 vtkTransform을 set해야 한다.

3) object node 생성

```
dotree::Object* object = dotree::Object::New();
object->SetName("obj_child");
object->SetActor( vtkActor );
```

잎노드인 dotree::Object를 생성하는 방법은 dotree::Object를 생성하고, SetName과 SetActor를 통해 속성을 설정해 주어야 한다.

4) Add child node (자식 노드 추가)

```
root->AddChild( transform );
transform->AddChild( object );
transform->AddChild( another_transform );
root->AddNode( "trans_child1", object );
```

trans_child1의 이름을 가진 노드 밑에 object노드를 add한다. 이때, object의 이름이 동일할 경우 추가 되지 않는다. root->AddNode() 형태로 써야 한다.

5) Remove child node (자식 노드만 삭제)

```
1  transform->RemoveChild( object );
2  transform->RemoveChild( another_transform );
3  root->RemoveChild( transform );
```

1라인은 transform 노드의 자식노드 중 object 노드를 삭제한다. object의 자손 노드가 있다면 그 자손들을 transform 노드의 자손으로 더한다. 2라인은 transform 노드의 자식 노드 중 another_transform 노드를 삭제한다. another_transform 노드의 자식 노드가 있다면 transform 노드에 자손 노드로 더한다. 3라인은 root 노드의 자식 노드 중 transform 노드를 삭제한다. transform 노드의 자식 노드가 있다면, root 노드에 자손들을 더한다. 특정 노드의 자식노드는 삭제할 수 있으나, 그 노드의 손자 노드는 바로 삭제할 수 없다. 먼저 삭제하고자 하는 손자노드의 부모노드를 찾은 다음, 그 부모노드로부터 삭제한다.

6) Remove children nodes

```
transform->RemoveChildren();
root->RemoveChildren();
```

RemoveChildren은 부모 아래의 모든 자식 노드 및 손자노드들까지 삭제한다.

7) Transform child node

```
1  transform->SetPosition( 2.0, 0.0, -2.0 );
2  transform->SetOrientation( 10, 0, 20 );
3  transform->SetScale( 2.0, 2.0, 2.0 );
4  transform->AddPosition( 2.0, 0.0, -2.0 );
5  transform->AddOrientation( 0, -90, 0 );
6  transform->AddScale( 2.0, 2.0, 2.0 );

7  double *xyz;
8  xyz = transform->GetPosition();
9  xyz = transform->GetOrientation();
10 xyz = transform->GetScale();
```

dotree::Transform 노드는 position과 orientation을 설정하는 함수를 제공하여, 자신의 모든 자손노드에 이를 반영한다. SetPosition(), SetOritentatio(), SetSca

le()함수는 절대적 변환을 AddPosition(), AddOrientation(), AddScale()은 상대적 변환을 수행한다. 1라인의 SetPosition()은 transform노드와 transform의 자손노드를 모두 (2.0, 0.0, −2.0)좌표로 이동하고, 4라인의 AddPosition()은 transform노드와 transform의 자손노드를 모두 현재 좌표에서(2.0, 0.0, −2.0) 만큼 이동한다. 7, 8, 9는 각각 transform 노드의 현재 해당 값들을 double[3]에 x, y, z 값을 포함하여 돌려 준다.

8) 자식 노드 개수

```
root->GetNumChildren();
transform->GetNumChildren();
```

자식을 가질 수 있는 dotree::Group 노드와 dotree::Tramsform 노드는 자식의 수를 가져올 수 있다.

9) 자식 노드와 관련된 기능들

```
 if (transform->ContainsNode( object ) == true())
{
     int index;
     index = transform->GetChildIndex( object );
}
```

이외에도 dotree::Transform과 dotree::Group등 자식노드를 가질 수 있는 클래스들은 자식노드의 존재 여부를 확인하는 node에 대한 포인터로 확인하는 ContainsNode()함수와 자식노드가 몇 번째 인지를 확인하는 GetChildIndex()함수도 제공한다.

10) 탐색 (이름으로 자손노드까지 탐색)

```
dotree::Node *node;

node = root->SearchNode("trans_child1");
node = transform->SearchNode("obj_child1");
```

dotree::Group과 dotree::Transform 클래스는 이름으로 자신의 자손노드를 찾을 수 있는 함수 SearchNode()를 제공한다.

11) 노드 이름 출력

```
std::string name;

name = root->GetName();
name = transform->GetName();
name = object->GetName();
```

dotree의 모든 타입의 노드들은 dotree::Node로부터 name을 상속받으므로 GetName() 함수를 통해 자신의 name을 가져 올 수 있다.

12) vtkActor list 가져오기

```
vtkActorCollection* ac = vtkActorCollection::New();
ac = root->GetActorList();
ac->InitTraversal();
unsigned int numOfActors = ac->GetNumberOfItems();
if( numOfActors > 0 )
{
   for( unsigned int i = 0; i < numOfActors; ++i )
   {
      vtkActor* actor = ac->GetNextActor();
      vtkActor->Delete();
   }
}
```

DOT에서 자식을 가질 수 있는 dotree::Group 과 dotree::Transform 은 자신의 자손인 vtkActor들의 리스트를 가져 올 수 있다. 이것은 GLOVE에서 이를 현재 scene에 그려진 vtkActor의 리스트를 가져올 수 있는 유일한 방법으로 Drawing Context를 유지하는 이유이기도 하다. 소스는 root아래의 모든 vtkActor를 메모리에서 지우는 작업을 수행한다.

5. 응용에 의존적인 Context 관리

3절에서 언급한 사용자의 오퍼레이션에 대한 history 관리는 매우 응용에 의존적인 context이다. GLOVE는 응용 맞춤형 사용자 인터페이스를 제공할 것이므로 응용마다 사용자가 사용할 수 있는 명령이나 이벤트의 종류가 달라질 것이기 때문이다. 또한, 화면에 그려지는 오브젝트들도 VTK 입장에서 생각하는 단위는 vtkActor일지 모르지만, 응용의 입장에서는 달라질 수 있다. 이를테면 로터 시뮬레이션 데이터의 경우, vtkActor는 각각 블레이드와 동체로 2개가 존재할 것이나, 헬기의 로터와 블레이드는 따로 움직일 수 없는 하나의 객체이다. 이러한 관점에

서 응용은 컴퓨터 그래픽스가 생각하는 context와 또 다른 의미의 context를 가진다고 할 수 있다. 따라서, 응용 맞춤형 사용자 인터페이스를 목표로 한다면, 응용 맞춤형 context의 관리가 필수적이다.

1) Command and Event History

GLOVE의 명령이나 이벤트의 히스토리 관리는 GLOVE 내부의 최근 수행한 command history list, event history list를 통해 관리한다.

2) Application Object Context

현재 scene에 표현되어 있는 오브젝트들의 리스트는 Drawing Object Tree를 검색하여 알 수 있다. 그러나, 매번 검색을 수행하는 것 보다 현재 scene 존재하는지 하지 않는지를 나타내는 오브젝트의 존재 여부에 대한 정보도 빠른 검색을 위해 존재할 필요가 있다. 범용 그래픽스 도구가 임의의 오브젝트에 대한 context를 관리해야 한다면, GLOVE와 같이 응용 맞춤형 사용자 인터페이스로 구성된 경우 scene 에 나타날 수 있는 오브젝트는 정해져 있다. 또한 그래픽스 관점의 오브젝트가 아니라 응용 관점의 의미론적 오브젝트 이므로 이는 응용에 맞게 재설계 될 수 있다. 단, 이들 오브젝트의 존재 여부를 나타낼 수 있는 추상화된 데이터 클래스만 존재하면, 응용별로 이를 상속받아 재구현할 수 있다. GLOVE는 SceneContext 클래스를 구현하여 이를 처리한다. SceneContext는 각 응용별로 다시 RotorSceneContext 등으로 확장될 수 있다.

6. 결론

본 문서에서는 하나의 scene을 만드는 context의 구성요소에 대해 정의하고, 이러한 scene을 관리하는 기존의 방법으로부터 GLOVE 시스템 context 관리방법을 도출하였다. GLOVE는 응용 맞춤형 사용자 인터페이스를 제공하는 환경으로 컴퓨터 그래픽스 관점의 scene 관리 뿐 아니라 응용 관점에서 필요한 context의 관리도 함께 요구한다. 따라서, 본 문서에서는 이러한 GLOVE 시스템에 맞는 Drawing Object Tree와 응용 관점의 context 관리 방법에 대해 제안하였다. Drawing Object Tree는 VRJuggler 와 vjVTK, VTK 환경에서 scene을 관리하는 최초의 데이터 구조이다. 향후, GLOVE가 확장됨에 따라 Drawing Object Tree는 잎노드의 확장과 context 관리를 위한 추가 정보를 위한 클래스 추가 등의 확장 개발이 필요하다.

7. 참고문헌

[1] Franclim Foping, "An overview of OpenSceneGraph", OSG Tutorial1, 2008

[2] Foping, S.F, "A polynomial algorithm to find the reversal degree of plan ar graphs, Master's thesis, 2006.

[3] Martz, P., "OpenSceneGraph QuickStart Guide", Skew Matrix Software, 2007.

[4] http://www.openscenegraph.org

GLOVE

0.1

Generated by Doxygen 1.6.1

Fri Dec 4 09:47:10 2009

Contents

1 Class Index **1**

1.1 Class Hierarchy 1

2 Class Index **3**

2.1 Class List 3

3 Class Documentation **5**

3.1 dotree::Group Class Reference 5

3.1.1 Detailed Description 7

3.1.2 Member Function Documentation 7

3.1.2.1 AddChild 7

3.1.2.2 AddNode 7

3.1.2.3 ContainsNode 7

3.1.2.4 FindActor 8

3.1.2.5 GetActorList 8

3.1.2.6 GetChild 8

3.1.2.7 GetChildIndex 8

3.1.2.8 GetGroup 9

3.1.2.9 GetNumChildren 9

3.1.2.10 InsertChild 9

3.1.2.11 RemoveChild 9

3.1.2.12 RemoveChildren 10

3.1.2.13 SearchNode 10

3.1.2.14 SetChild 10

3.2 dotree::Node Class Reference 11

3.2.1 Detailed Description 12

3.2.2 Constructor & Destructor Documentation 12
3.2.2.1 ~Node 12
3.2.3 Member Function Documentation 12
3.2.3.1 AddParent 12
3.2.3.2 GetGroup 13
3.2.3.3 GetName 13
3.2.3.4 GetObject 13
3.2.3.5 GetParent 13
3.2.3.6 GetTransform 13
3.2.3.7 RemoveParent 14
3.2.3.8 SetName 14
3.2.3.9 SetName 14
3.3 dotree::Object Class Reference 15
3.3.1 Detailed Description 16
3.3.2 Member Function Documentation 16
3.3.2.1 GetActor 16
3.3.2.2 GetPanel 16
3.3.2.3 SetActor 16
3.3.2.4 SetPanel 17
3.4 dotree::Transform Class Reference 18
3.4.1 Detailed Description 20
3.4.2 Member Function Documentation 20
3.4.2.1 AddOrientation 20
3.4.2.2 AddOrientation 20
3.4.2.3 AddPosition 20
3.4.2.4 AddPosition 20
3.4.2.5 AddScale 21
3.4.2.6 AddScale 21
3.4.2.7 GetOrientation 21
3.4.2.8 GetPosition 21
3.4.2.9 GetScale 21
3.4.2.10 GetVTKTransform 21
3.4.2.11 SetOrientation 22
3.4.2.12 SetOrientation 22

3.4.2.13 SetPosition 22
3.4.2.14 SetPosition 22
3.4.2.15 SetScale 22
3.4.2.16 SetScale 23
3.4.2.17 SetTransform 23

Chapter 1

Class Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

dotree::Node 11
dotree::Group 5
dotree::Transform 18
dotree::Object 15

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

dotree::Group (**Group** (p. 5) class is base class for any node that can have children. It is key class in the spatial organization of Tree) 5
dotree::Node (**Node** (p. 11) is a base class for all nodes in the tree and store the parent of node) 11
dotree::Object (**Object** (p. 15) class corresponds to the leaf node in dotree. It contains vtkActor) 15
dotree::Transform (**Transform** (p. 18) class contains vtkTransform that transforms the geometry of its children, allowing scene objects to be rotated, translated and scaled) 18

Chapter 3

Class Documentation

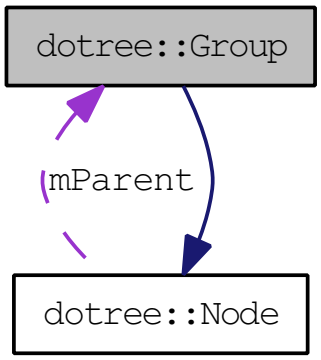
3.1 dotree::Group Class Reference

Group (p. 5) class is base class for any node that can have children. It is key class in the spatial organization of Tree.

`#include <glvGroup.h>`

Inherits **dotree::Node**.

Inherited by **dotree::Transform**.Collaboration diagram for dotree::Group:



Public Member Functions

- ~**Group** ()

  Destructor.

- virtual **Group** ∗ **GetGroup** ()
- virtual bool **AddNode** (const char ∗name, **Node** ∗child)

  *Add **Node** (p. 11) to **Group** (p. 5) using root.*

- virtual bool **AddChild** (**Node** *child)

  *Add **Node** (p. 11) to **Group** (p. 5).*

- virtual bool **InsertChild** (unsigned int index, **Node** *child)

  *Insert **Node** (p. 11) to **Group** (p. 5) at specific location.*

- virtual bool **RemoveChild** (**Node** *child)

  *Remove a **Node** (p. 11) from **Group** (p. 5).*

- virtual bool **RemoveChildren** ()

  *Remove Children from **Group** (p. 5).*

- unsigned int **GetNumChildren** () const
- virtual bool **SetChild** (unsigned int i, **Node** *node)

  Set child node at position i.

- **Node** * **GetChild** (unsigned int i)
- bool **ContainsNode** (const **Node** *node) const
- unsigned int **GetChildIndex** (const **Node** *node) const
- virtual **Node** * **SearchNode** (std::string searchName)

  Search the node by name.

- virtual void **FindActor** (vtkActorCollection *ac)

  Find the vtkActor and Added in vtkActor list.

- virtual vtkActorCollection * **GetActorList** ()

  Get the vtkActor list.

- virtual void **Print** ()

  Print the nodes by tree structure.

Static Public Member Functions

- static **Group** * **New** ()

  Constructor.

Protected Member Functions

- **Group** ()

  Constructor.

Protected Attributes

- NodeList **mChildren**

  Child node list.

- vtkActorCollection * **mActorCol**

  vtkActor list

3.1.1 Detailed Description

Group (p. 5) class is base class for any node that can have children. It is key class in the spatial organization of Tree.

3.1.2 Member Function Documentation

3.1.2.1 bool Group::AddChild (Node * *child*) [virtual]

Add **Node** (p. 11) to **Group** (p. 5). Add **Node** (p. 11) to **Group** (p. 5)

Parameters:

child node

Returns:

true for success, otherwise return false

3.1.2.2 bool Group::AddNode (const char * *name*, Node * *child*) [virtual]

Add **Node** (p. 11) to **Group** (p. 5) using root. Add **Node** (p. 11) to **Group** (p. 5) using root

Parameters:

parent node name, child node

Returns:

true for success, otherwise return false

3.1.2.3 bool dotree::Group::ContainsNode (const Node * *node*) const [inline]

Return true if node is contained within **Group** (p. 5)

Parameters:

child node

Returns:

true for success, otherwise return false

3.1.2.4 void Group::FindActor (vtkActorCollection * *ac*) [virtual]

Find the vtkActor and Added in vtkActor list. Find the vtkActor and added in vtkActor list

Parameters:

vtkActorCollection

3.1.2.5 vtkActorCollection * Group::GetActorList () [virtual]

Get the vtkActor list. Get the vtkActor list

Returns:

actorlist

3.1.2.6 Node* dotree::Group::GetChild (unsigned int *i*) [inline]

Return child node at position i

Parameters:

specified position i

Returns:

child node

3.1.2.7 unsigned int dotree::Group::GetChildIndex (const Node * *node*) const [inline]

Get the index number of child Return a value between 0 and mChildren.size()-1

Parameters:

child node

Returns:

the index number of child, if not found then return mChildren.size()

3.1.2.8 virtual Group* dotree::Group::GetGroup () [inline, virtual]

Get the **Group** (p. 5) pointer

Returns:

Group (p. 5) pointer if succeeds, 0 if fails

Reimplemented from **dotree::Node** (p. 13).

3.1.2.9 unsigned int dotree::Group::GetNumChildren () const [inline]

Return the number of children nodes

Returns:

the number of children nodes

3.1.2.10 bool Group::InsertChild (unsigned int *index*, Node * *child*) [virtual]

Insert **Node** (p. 11) to **Group** (p. 5) at specific location. Insert **Node** (p. 11) to **Group** (p. 5) at specific location

Parameters:

specified index, child node

Returns:

true for success, otherwise return false

3.1.2.11 bool Group::RemoveChild (Node * *child*) [virtual]

Remove a **Node** (p. 11) from **Group** (p. 5). Remove a **Node** (p. 11) from **Group** (p. 5)

Parameters:

start position, number of children to remove

Returns:

true for success, otherwise return false

3.1.2.12 bool Group::RemoveChildren () [virtual]

Remove Children from **Group** (p. 5). Remove Nodes from **Group** (p. 5) If **Node** (p. 11) is contained in **Group** (p. 5) then remove it from the child list

Parameters:

child node

Returns:

true for sucesss, otherwise return false

3.1.2.13 Node * Group::SearchNode (std::string *searchName*) [virtual]

Search the node by name. Search the node by name

Parameters:

string

Returns:

node

3.1.2.14 bool Group::SetChild (unsigned int *i*, Node * *node*) [virtual]

Set child node at position i. Set child node at position i

Parameters:

specified position, child node

Returns:

true for success, otherwise return false

The documentation for this class was generated from the following files:

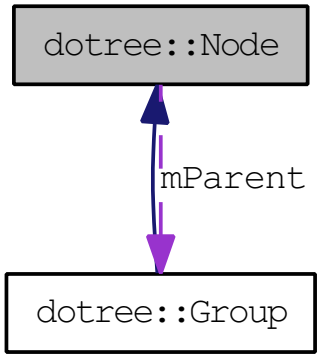
- /home/osung/tmp/glove/trunk/include/dotree/glvGroup.h
- /home/osung/tmp/glove/trunk/src/dotree/glvGroup.cc

3.2 dotree::Node Class Reference

Node (p. 11) is a base class for all nodes in the tree and store the parent of node.

`#include <glvNode.h>`

Inherited by **dotree::Group**, and **dotree::Object**.Collaboration diagram for dotree::Node:



Public Member Functions

- ~**Node** ()
  Destructor.
- **Group** * **GetParent** ()
- void **SetName** (std::string name)
- void **SetName** (char *name)
- std::string & **GetName** ()
- virtual **Group** * **GetGroup** ()
- virtual **Transform** * **GetTransform** ()
- virtual **Object** * **GetObject** ()

Static Public Member Functions

- static **Node** * **New** ()
  Constructor.

Protected Member Functions

- **Node** ()
  Constructor.

- void **AddParent** (**dotree::Group** *node)

  Add the group node (parent).

- void **RemoveParent** (**dotree::Group** *node)

  Remove the group node (parent).

Protected Attributes

- **Group** * **mParent**

  dotree::Group *(p. 5) pointers which is used to store the parent(s) of node*

- std::string **mName**

  name

Friends

- class **dotree::Group**

  full access to the private data members of ***Group*** *(p. 5) class*

3.2.1 Detailed Description

Node (p. 11) is a base class for all nodes in the tree and store the parent of node.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 Node::~Node ()

Destructor. Destuctor.

3.2.3 Member Function Documentation

3.2.3.1 void Node::AddParent (dotree::Group * *node*) [protected]

Add the group node (parent). Add Parent

Parameters:

parent node

3.2.3.2 virtual Group* dotree::Node::GetGroup () [inline, virtual]

Convert 'this' into a **Group** (p. 5) pointer if **Node** (p. 11) is a **Group** (p. 5), otherwise return 0

Returns:

Group (p. 5) pointer if succeeds, 0 if fails

Reimplemented in **dotree::Group** (p. 9).

3.2.3.3 std::string& dotree::Node::GetName () [inline]

Get the name of **Node** (p. 11)

Returns:

name of **Node** (p. 11)

3.2.3.4 virtual Object* dotree::Node::GetObject () [inline, virtual]

Convert 'this' into a **Object** (p. 15) pointer if **Node** (p. 11) is a **Object** (p. 15), otherwise return 0

Returns:

Object (p. 15) pointer if succeeds, 0 if fails

Reimplemented in **dotree::Object** (p. 15).

3.2.3.5 Group* dotree::Node::GetParent () [inline]

Get a parent of node

Returns:

the parent of node

3.2.3.6 virtual Transform* dotree::Node::GetTransform () [inline, virtual]

Convert 'this' into a **Transform** (p. 18) pointer if **Node** (p. 11) is a **Transform** (p. 18), otherwise return 0

Returns:

Transform (p. 18) pointer if succeeds, 0 if fails

Reimplemented in **dotree::Transform** (p. 18).

3.2.3.7 void Node::RemoveParent (dotree::Group * *node*) [protected]

Remove the group node (parent). Remove Parent

Parameters:

parent node

3.2.3.8 void dotree::Node::SetName (char * *name*) [inline]

Set the name of **Node** (p. 11)

Parameters:

name of **Node** (p. 11)

3.2.3.9 void dotree::Node::SetName (std::string *name*) [inline]

Set the name of **Node** (p. 11)

Parameters:

name of **Node** (p. 11)

The documentation for this class was generated from the following files:

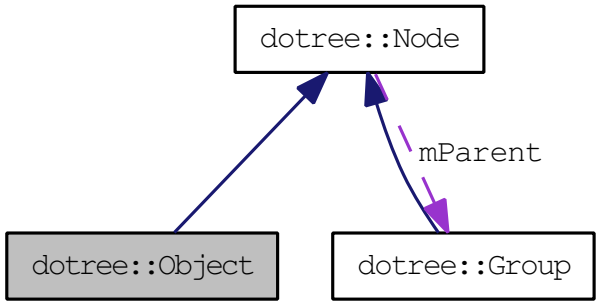
- /home/osung/tmp/glove/trunk/include/dotree/glvNode.h
- /home/osung/tmp/glove/trunk/src/dotree/glvNode.cc

3.3 dotree::Object Class Reference

Object (p. 15) class corresponds to the leaf node in dotree. It contains vtkActor.

`#include <glvObject.h>`

Inherits **dotree::Node**.Collaboration diagram for dotree::Object:



Public Member Functions

- ~**Object** ()
  Destructor.
- virtual **Object** * **GetObject** ()
  *Get the **Object** (p. 15) pointer if it is object node.*
- virtual bool **SetActor** (vtkActor *actor)
  Set the vtkActor.
- vtkActor * **GetActor** ()
- virtual bool **SetPanel** (vjPanelWidget *panel)
  Set the vjPanelWidget.
- vjPanelWidget * **GetPanel** ()

Static Public Member Functions

- static **Object** * **New** ()
  Constructor.

Protected Member Functions

- **Object** ()
  Constructor.

Protected Attributes

- vtkActor * **mActor**

  vtkActor

- vjPanelWidget * **mPanel**

  vjPanelWidget

3.3.1 Detailed Description

Object (p. 15) class corresponds to the leaf node in dotree. It contains vtkActor.

3.3.2 Member Function Documentation

3.3.2.1 vtkActor* dotree::Object::GetActor () [inline]

Get the vtkActor

Returns:

vtkActor

3.3.2.2 vjPanelWidget* dotree::Object::GetPanel () [inline]

Get the vjPanelWidget

Returns:

vjPanelWidget

3.3.2.3 bool Object::SetActor (vtkActor * *actor*) [virtual]

Set the vtkActor. Set the vtkActor

Parameters:

vtkActor

Returns:

true for success, otherwise return false

3.3.2.4 bool Object::SetPanel (vjPanelWidget * *panel*) [virtual]

Set the vjPanelWidget. Set the vjPanelWidget

Parameters:

vjPanelWidget

Returns:

true for success, otherwise return false

The documentation for this class was generated from the following files:

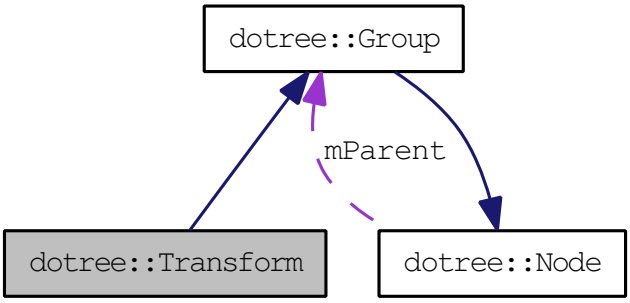
- /home/osung/tmp/glove/trunk/include/dotree/glvObject.h
- /home/osung/tmp/glove/trunk/src/dotree/glvObject.cc

3.4 dotree::Transform Class Reference

Transform (p. 18) class contains vtkTransform that transforms the geometry of its children, allowing scene objects to be rotated, translated and scaled.

`#include <glvTransform.h>`

Inherits **dotree::Group**.Collaboration diagram for dotree::Transform:



Public Member Functions

- ~**Transform** ()
  Destructor.
- virtual **Transform** * **GetTransform** ()
  Get the transform pointer if it is transform node.
- virtual void **Decompose** ()
  Decompose transform matrix into its various value.
- virtual void **SetPosition** (double *position)
  Set the position.
- virtual void **SetPosition** (double x, double y, double z)
  Set the position.
- const double * **GetPosition** ()
- virtual void **AddPosition** (double *position)
  Add the position.
- virtual void **AddPosition** (double x, double y, double z)
  Add the position.
- virtual void **SetOrientation** (double *orientation)
  Set the orientation.

- virtual void **SetOrientation** (double x, double y, double z)
  Set the orientation.
- const double * **GetOrientation** ()
- virtual void **AddOrientation** (double *orientation)
  Add the orientation.
- virtual void **AddOrientation** (double x, double y, double z)
  Add the orientation.
- virtual void **SetScale** (double *scale)
  Set the scale.
- virtual void **SetScale** (double x, double y, double z)
  Set the scale.
- const double * **GetScale** ()
- virtual void **AddScale** (double *scale)
  Add the scale.
- virtual void **AddScale** (double x, double y, double z)
  Add the scale.
- virtual bool **SetTransform** (vtkTransform *transform)
  Set the vtkTransform.
- vtkTransform * **GetVTKTransform** ()

Static Public Member Functions

- static **Transform** * **New** ()
  Constructor.

Protected Member Functions

- **Transform** ()
  Constructor.

Protected Attributes

- double **mPos** [3]
  Transform *(*p. 18*) variables.*

- vtkTransform * **mTransform**

  vtkTransform

3.4.1 Detailed Description

Transform (p. 18) class contains vtkTransform that transforms the geometry of its children, allowing scene objects to be rotated, translated and scaled.

3.4.2 Member Function Documentation

3.4.2.1 void Transform::AddOrientation (double *x*, double *y*, double *z*) [virtual]

Add the orientation. Add the orientation

Parameters:

x,y,z

3.4.2.2 void Transform::AddOrientation (double * *orientation*) [virtual]

Add the orientation. Add the orientation

Returns:

rotation pointer

3.4.2.3 void Transform::AddPosition (double *x*, double *y*, double *z*) [virtual]

Add the position. Add the position

Parameters:

x,y,z

3.4.2.4 void Transform::AddPosition (double * *position*) [virtual]

Add the position. Add the position

Parameters:

position pointer

3.4.2.5 void Transform::AddScale (double *x*, double *y*, double *z*) [virtual]

Add the scale. Add the scale

Parameters:

x,y,z

3.4.2.6 void Transform::AddScale (double * *scale*) [virtual]

Add the scale. Add the scale

Parameters:

scale pointer

3.4.2.7 const double* dotree::Transform::GetOrientation () [inline]

Get the orientation

Returns:

rotation value

3.4.2.8 const double* dotree::Transform::GetPosition () [inline]

Get the position

Returns:

position value

3.4.2.9 const double* dotree::Transform::GetScale () [inline]

Get the scale

Returns:

scale value

3.4.2.10 vtkTransform* dotree::Transform::GetVTKTransform () [inline]

Get the vtkTransform

Returns:

vtkTransform

3.4.2.11 void Transform::SetOrientation (double *x*, double *y*, double *z*) [virtual]

Set the orientation. Set the orientation

Parameters:

x,y,z

3.4.2.12 void Transform::SetOrientation (double * *orientation*) [virtual]

Set the orientation. Set the orientation

Parameters:

rotation pointer

3.4.2.13 void Transform::SetPosition (double *x*, double *y*, double *z*) [virtual]

Set the position. Set the position

Parameters:

x,y,z

3.4.2.14 void Transform::SetPosition (double * *position*) [virtual]

Set the position. Set the position

Parameters:

position pointer

3.4.2.15 void Transform::SetScale (double *x*, double *y*, double *z*) [virtual]

Set the scale. Set the scale

Parameters:

x,y,z

3.4.2.16 void Transform::SetScale (double * *scale*) [virtual]

Set the scale. Set the scale

Parameters:

scale pointer

3.4.2.17 bool Transform::SetTransform (vtkTransform * *transform*) [virtual]

Set the vtkTransform. Set the vtkTransform

Parameters:

vtkTransform

Returns:

true for success, otherwise return false

The documentation for this class was generated from the following files:

- /home/osung/tmp/glove/trunk/include/dotree/glvTransform.h
- /home/osung/tmp/glove/trunk/src/dotree/glvTransform.cc

Index

~Node
    dotree::Node, 12

AddChild
    dotree::Group, 7
AddNode
    dotree::Group, 7
AddOrientation
    dotree::Transform, 20
AddParent
    dotree::Node, 12
AddPosition
    dotree::Transform, 20
AddScale
    dotree::Transform, 20, 21

ContainsNode
    dotree::Group, 7

dotree::Group, 5
    AddChild, 7
    AddNode, 7
    ContainsNode, 7
    FindActor, 8
    GetActorList, 8
    GetChild, 8
    GetChildIndex, 8
    GetGroup, 8
    GetNumChildren, 9
    InsertChild, 9
    RemoveChild, 9
    RemoveChildren, 9
    SearchNode, 10
    SetChild, 10
dotree::Node, 11
    ~Node, 12
    AddParent, 12
    GetGroup, 12
    GetName, 13
    GetObject, 13
    GetParent, 13
    GetTransform, 13
    RemoveParent, 13
    SetName, 14
dotree::Object, 15
    GetActor, 16
    GetPanel, 16
    SetActor, 16
    SetPanel, 16
dotree::Transform, 18
    AddOrientation, 20
    AddPosition, 20
    AddScale, 20, 21
    GetOrientation, 21
    GetPosition, 21
    GetScale, 21
    GetVTKTransform, 21
    SetOrientation, 21, 22
    SetPosition, 22
    SetScale, 22
    SetTransform, 23

FindActor
    dotree::Group, 8

GetActor
    dotree::Object, 16
GetActorList
    dotree::Group, 8
GetChild
    dotree::Group, 8
GetChildIndex
    dotree::Group, 8
GetGroup
    dotree::Group, 8
    dotree::Node, 12
GetName
    dotree::Node, 13
GetNumChildren
    dotree::Group, 9
GetObject
    dotree::Node, 13

GetOrientation
dotree::Transform, 21
GetPanel
dotree::Object, 16
GetParent
dotree::Node, 13
GetPosition
dotree::Transform, 21
GetScale
dotree::Transform, 21
GetTransform
dotree::Node, 13
GetVTKTransform
dotree::Transform, 21

InsertChild
dotree::Group, 9

RemoveChild
dotree::Group, 9
RemoveChildren
dotree::Group, 9
RemoveParent
dotree::Node, 13

SearchNode
dotree::Group, 10
SetActor
dotree::Object, 16
SetChild
dotree::Group, 10
SetName
dotree::Node, 14
SetOrientation
dotree::Transform, 21, 22
SetPanel
dotree::Object, 16
SetPosition
dotree::Transform, 22
SetScale
dotree::Transform, 22
SetTransform
dotree::Transform, 23