CS 380

Introduction to Computer Graphics

LAB (5)

2018.04.09

Preliminaries

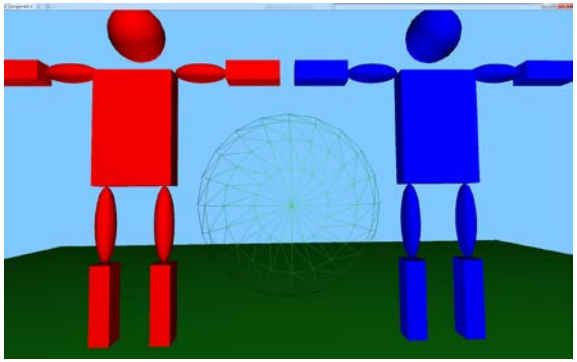
KAIST

- You would be better to get familiar with the following concepts to complete this assignment:
  - Class
  - Vector (stl)
  - Pointer
- There are many classes and functions in this homework, **so please read the description (pdf and code) in advance**, and start to do some coding.

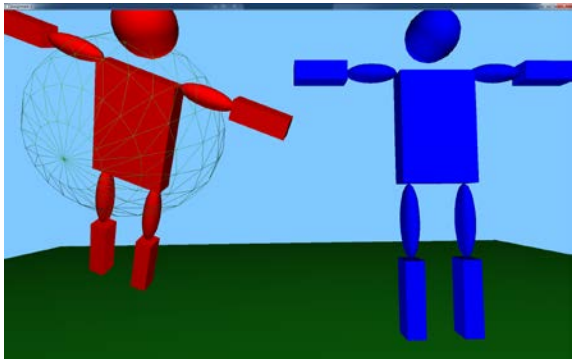
Goals

KAIST

- Scene graph
  - Build a structure for dealing with objects in a smart way



- Picking
  - Implement user manipulation code



- **Note that you have to read the description file (pdf) and the detailed description in the code thoroughly**

Task 1. Code Migration

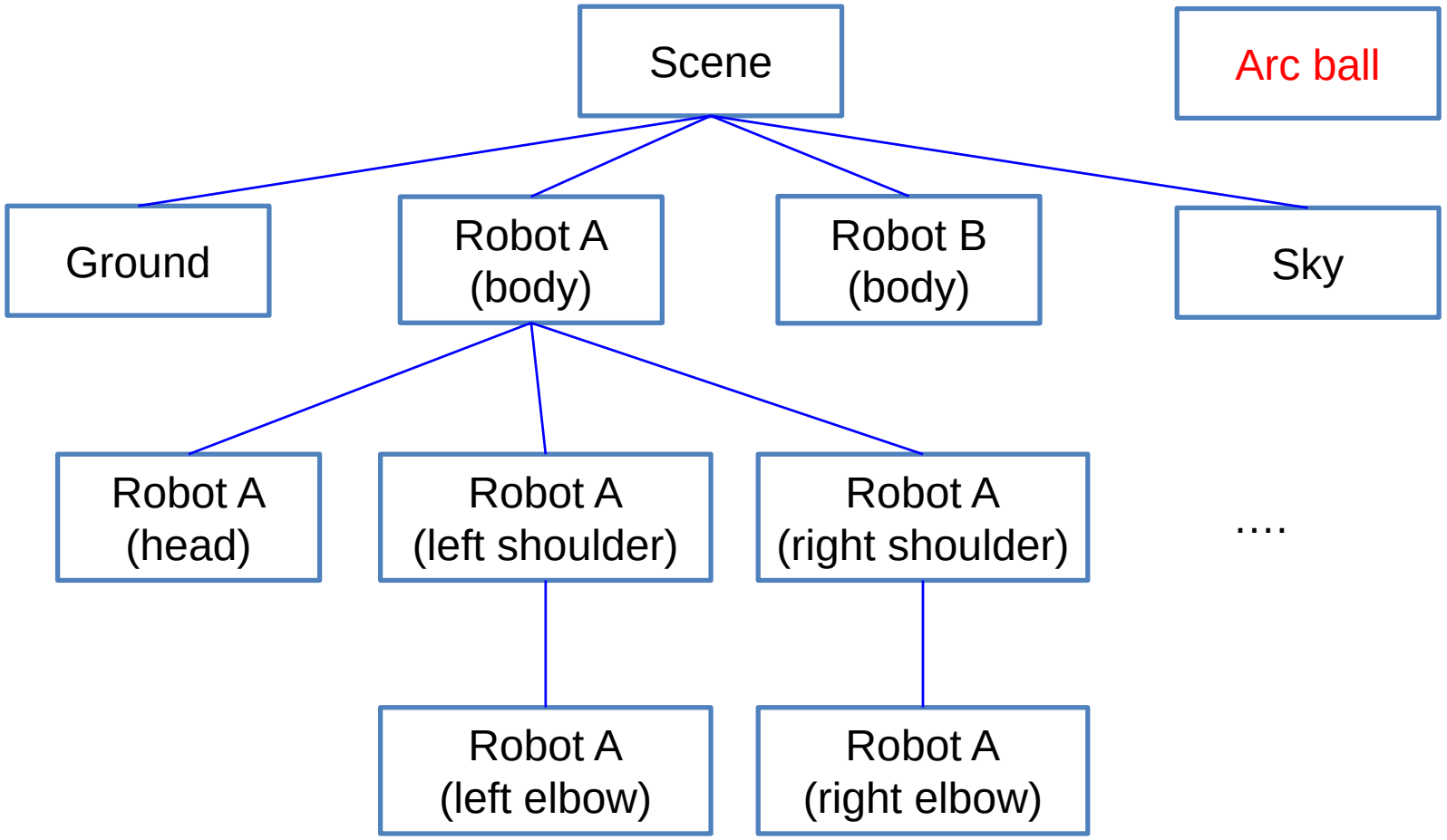
KAIST

- In this project, asst4 is based on your asst3.
  - You should finish previous homework before start.
- Make the copy of asst3 project (named asst4) and add asst4.zip file into your new project.
- ‘asst4-snippets.cpp’ can help you for modifying your asst4.cpp code
  - Construct the scene graph
  - Draw the scene graph

Scene Graph

KAIST

- Whole scene = 1 tree structure



VISUAL COMPUTING Lab

Scene Graph

KAIST

- Two kinds of nodes on scene graph
  - Transform nodes
    - RBT with respect to its parent frame

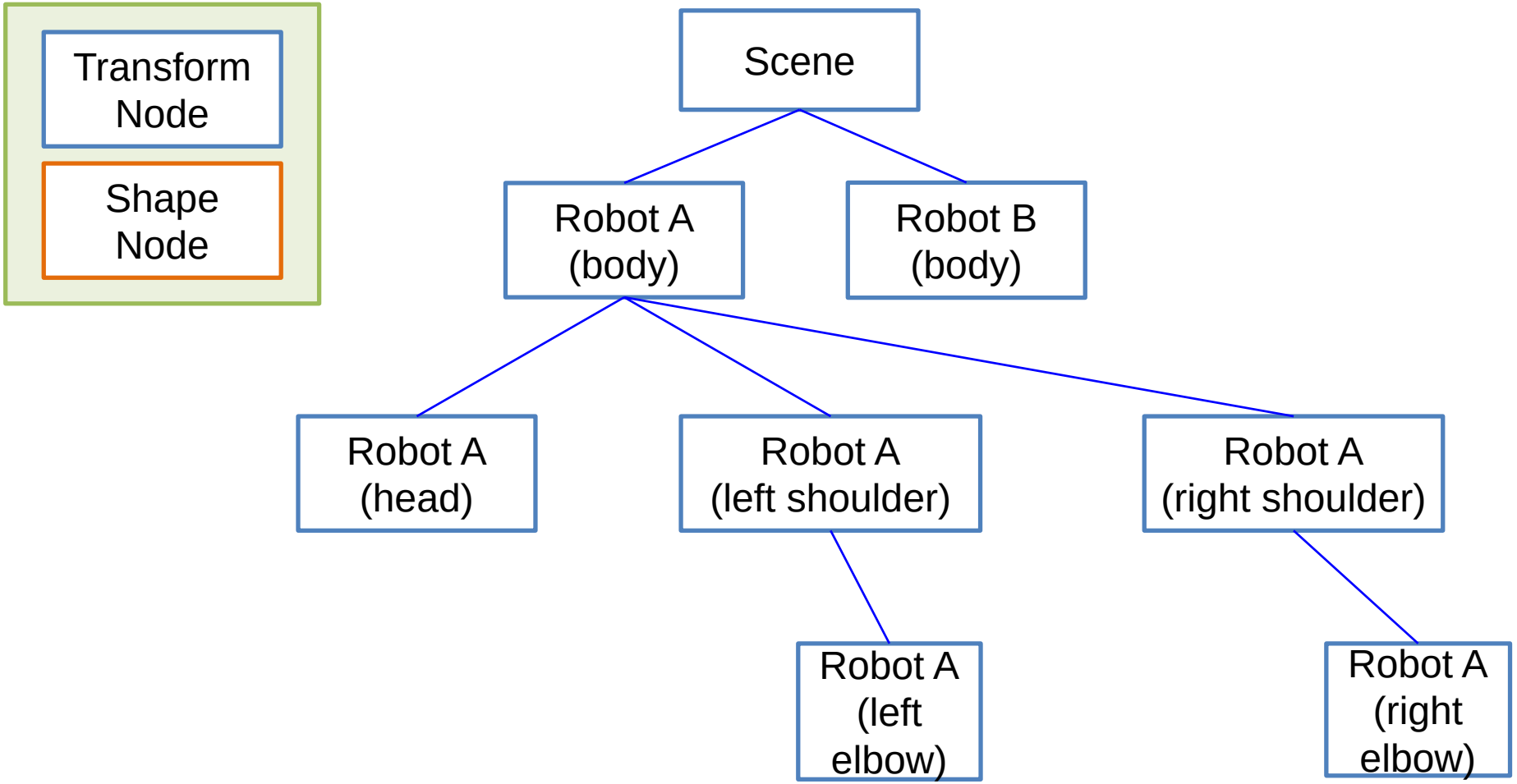
$$\begin{aligned} \vec{\mathbf{o}}^t &= \vec{\mathbf{w}}^t O \\ \vec{\mathbf{s}}^t &= \vec{\mathbf{o}}^t S \\ \vec{\mathbf{l}}^t &= \vec{\mathbf{s}}^t L \end{aligned}$$

  - Shape nodes
    - Matrix4 (AffineMatrix) : geometry to be drawn

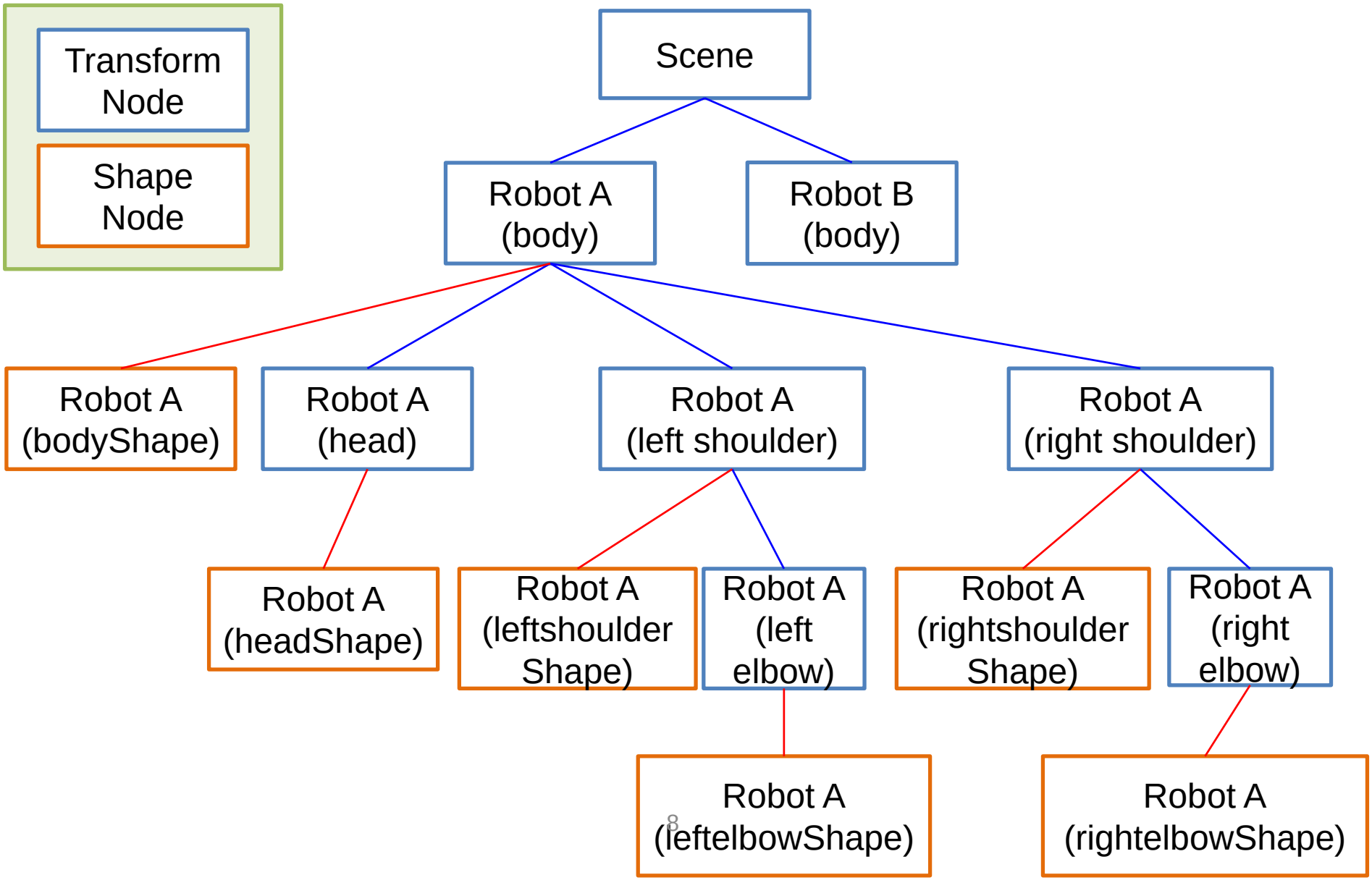
$$\vec{\mathbf{b}}^t = \vec{\mathbf{l}}^t B = \vec{\mathbf{l}}^t \cdot \mathrm{Trans} \cdot \mathrm{Scale}$$

Scene Graph

KAIST



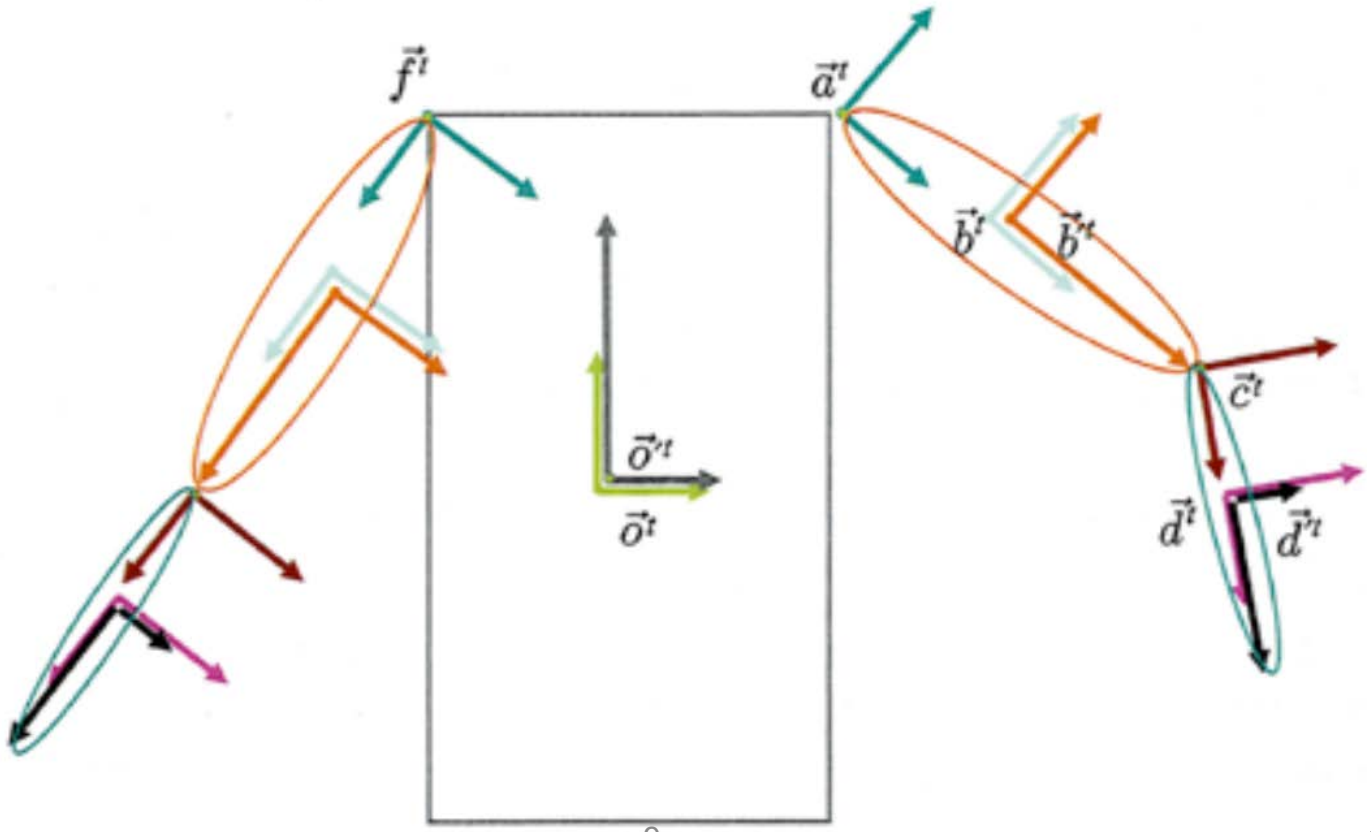
Scene Graph


Transform Node
Shape Node
Scene
Robot A (body)
Robot B (body)
Robot A (bodyShape)
Robot A (head)
Robot A (left shoulder)
Robot A (right shoulder)
Robot A (headShape)
Robot A (leftshoulder Shape)
Robot A (left elbow)
Robot A (rightshoulder Shape)
Robot A (right elbow)
Robot A (leftelbowShape)
Robot A (rightelbowShape)

Scene graph

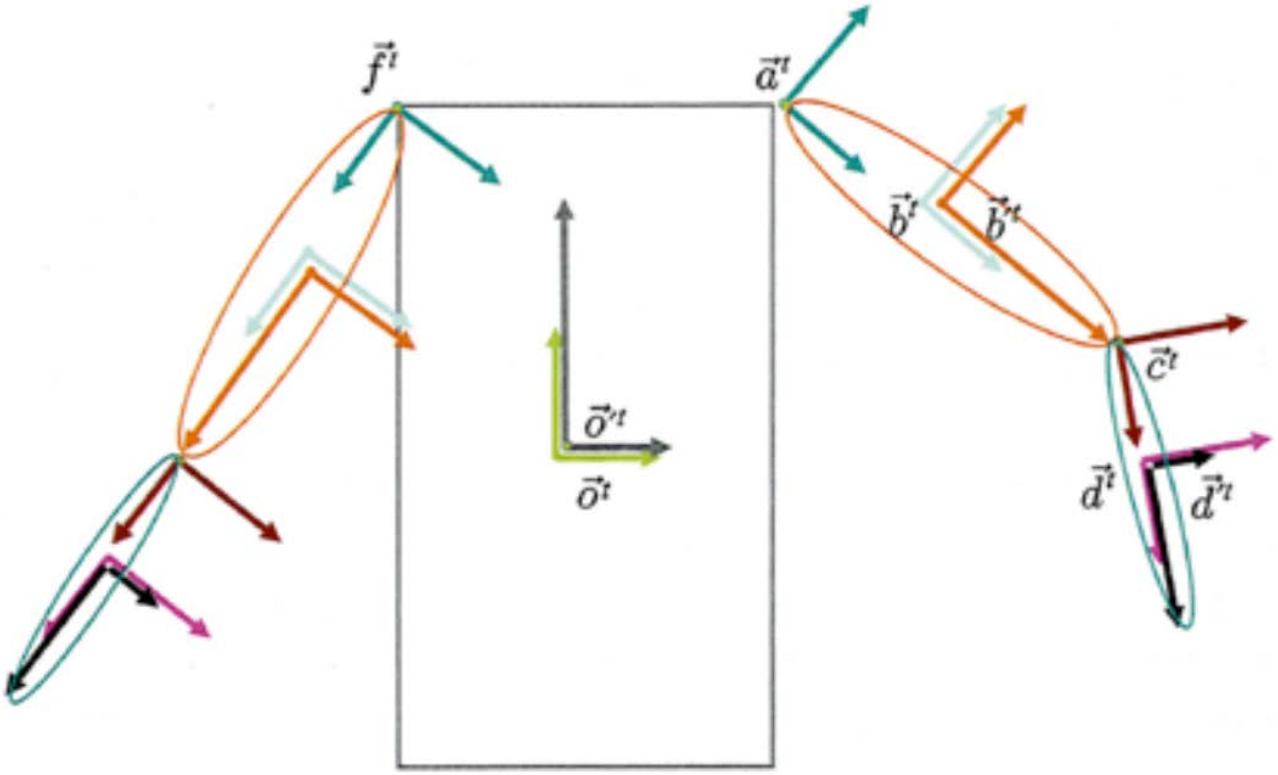
KAIST

- If we move the body of the robot, the connected components should be modified automatically.



Scene graph

- In order to build it, we describe an object frame with the previous object frame, not the world frame.

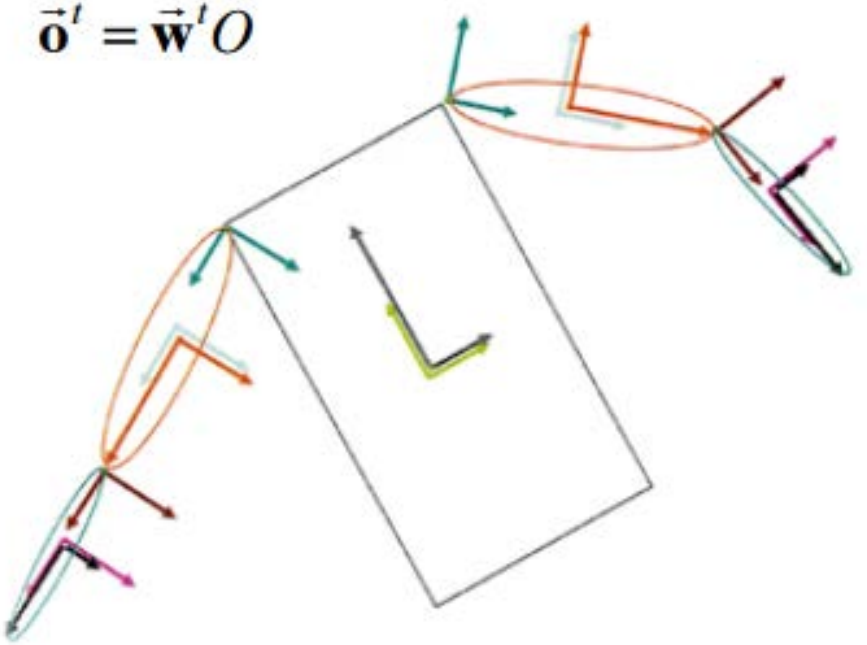


$$\vec{\mathbf{o}}^t = \vec{\mathbf{w}}^t O$$
$$\vec{\mathbf{o}}'^t = \vec{\mathbf{o}}^t O'$$
$$\vec{\mathbf{a}}^t = \vec{\mathbf{o}}^t A$$
$$\vec{\mathbf{b}}^t = \vec{\mathbf{a}}^t B$$
$$\vec{\mathbf{b}}'^t = \vec{\mathbf{b}}^t B'$$
$$\vec{\mathbf{c}}^t = \vec{\mathbf{b}}^t C$$
$$\vec{\mathbf{d}}^t = \vec{\mathbf{c}}^t D$$
$$\vec{\mathbf{d}}'^t = \vec{\mathbf{d}}^t D'$$
$$\vec{\mathbf{f}}^t = \vec{\mathbf{o}}^t F$$

$$\vec{d}'^t c_{d'} = \vec{w}^t OABCDD' c_{d'}$$

Scene graph

- Other parts would be changed following the scene graph when we modify the object frame $\vec{\mathbf{o}}^t = \vec{\mathbf{w}}^t O$.



$$\vec{\mathbf{o}}^t = \vec{\mathbf{w}}^t O$$
$$\vec{\mathbf{o}}'^t = \vec{\mathbf{o}}^t O'$$
$$\vec{\mathbf{a}}^t = \vec{\mathbf{o}}^t A$$
$$\vec{\mathbf{b}}^t = \vec{\mathbf{a}}^t B$$
$$\vec{\mathbf{b}}'^t = \vec{\mathbf{b}}^t B'$$
$$\vec{\mathbf{c}}^t = \vec{\mathbf{b}}^t C$$
$$\vec{\mathbf{d}}^t = \vec{\mathbf{c}}^t D$$
$$\vec{\mathbf{d}}'^t = \vec{\mathbf{d}}^t D'$$
$$\vec{\mathbf{f}}^t = \vec{\mathbf{o}}^t F$$

Visitor (SgNodeVisitor)

KAIST

- Class for easy traversal on the scene graph
- We need to do various operations through the nodes of scene graph

```
class SgNodeVisitor {
public:
  virtual bool visit(SgTransformNode& node);
  virtual bool visit(SgShapeNode& node);

  virtual bool postVisit(SgTransformNode& node);
  virtual bool postVisit(SgShapeNode& node);
};
```

Visitor (SgNodeVisitor)

KAIST

- There are three types of visitor
  - Drawer, Picker and RbtAccumVisitor
  - Each visitor has different role, variables and functions (visit( ), postVisit( ), ..)

```
class Drawer : public SgNodeVisitor {
protected:
  std::vector<RigTForm> rbtStack_;
```

```
class Picker : public SgNodeVisitor {
  std::vector<std::tr1::shared_ptr<SgNode> > nodeStack_;
```

```
class RbtAccumVisitor : public SgNodeVisitor {
protected:
    vector<RigTForm> rbtStack_;
```

accept( ) and visit( )

KAIST

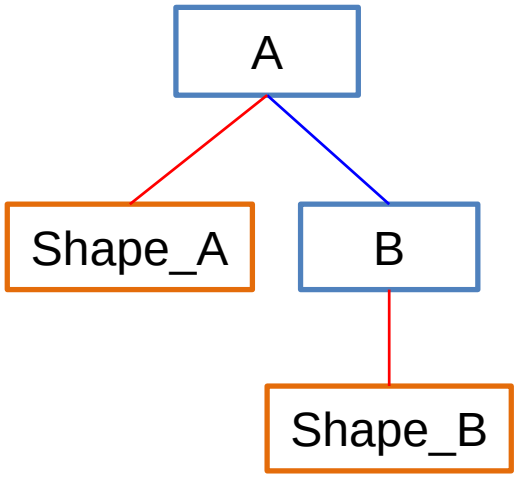
Each node has accept( )

: apply visit( ) for itself (node) and

pass the visitor to its children

node's accept() function

```
bool SgTransformNode::accept(SgNodeVisitor& visitor) {
  if (!visitor.visit(*this))
    return false;
  for (int i = 0, n = children_.size(); i < n; ++i) {
    if (!children_[i]->accept(visitor))
      return false;
  }
  return visitor.postVisit(*this);
}
```



Each visitor has visit( )

: visit( ) do some works on certain node

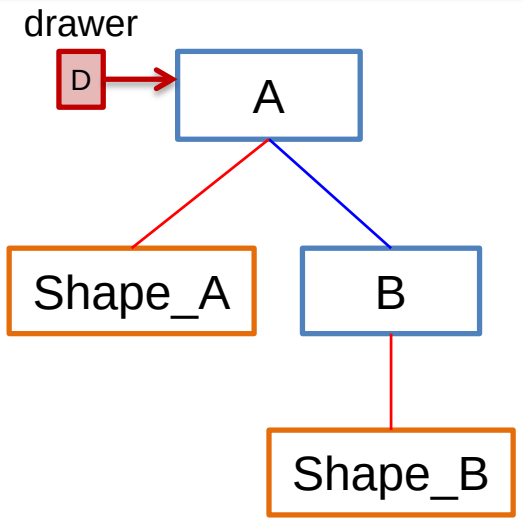
accept( ) and visit( )

KAIST

Example

A->accept(drawer)

Operation



drawer's visit() function

```
virtual bool visit(SgTransformNode& node) {
  rbtStack_.push_back(rbtStack_.back( ) * node.getRbt( ));
  return true;
}

virtual bool visit(SgShapeNode& shapeNode) {
  const Matrix4 MVM = rigTFormToMatrix(rbtStack_.back( )) * shapeNode.getAffineMatrix( );
  sendModelViewNormalMatrix(curSS_, MVM, normalMatrix(MVM));
  shapeNode.draw(curSS_);
  return true;
}
```

accept( ) and visit( ) for drawer

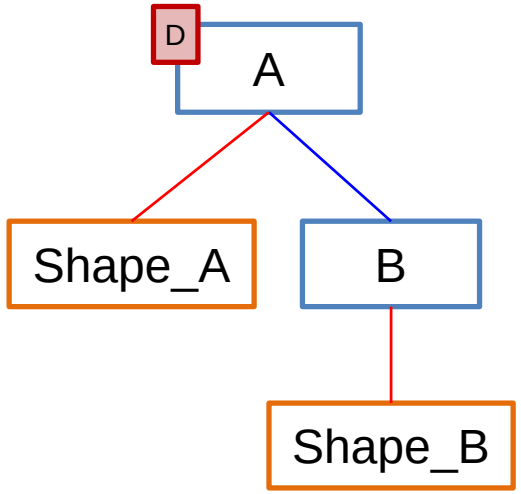
KAIST

Example

A->accept(drawer)

drawer.visit(A)

Operation



Stack = {A}

drawer's visit() function

```
virtual bool visit(SgTransformNode& node) {
  rbtStack_.push_back(rbtStack_.back( ) * node.getRbt( ));
  return true;
}

virtual bool visit(SgShapeNode& shapeNode) {
  const Matrix4 MVM = rigTFormToMatrix(rbtStack_.back( )) * shapeNode.getAffineMatrix( );
  sendModelViewNormalMatrix(curSS_, MVM, normalMatrix(MVM));
  shapeNode.draw(curSS_);
  return true;
}
```

accept( ) and visit( ) for drawer

KAIST

Example

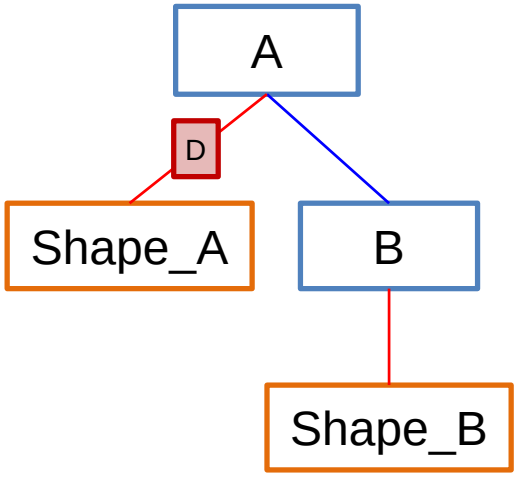
A->accept(drawer)

drawer.visit(A)

pass ‘drawer’ to Shape_A

Operation

Stack = {A}



drawer’s visit() function

```
virtual bool visit(SgTransformNode& node) {
  rbtStack_.push_back(rbtStack_.back( ) * node.getRbt( ));
  return true;
}

virtual bool visit(SgShapeNode& shapeNode) {
  const Matrix4 MVM = rigTFormToMatrix(rbtStack_.back( )) * shapeNode.getAffineMatrix( );
  sendModelViewNormalMatrix(curSS_, MVM, normalMatrix(MVM));
  shapeNode.draw(curSS_);
  return true;
}
```

accept( ) and visit( ) for drawer

KAIST

Example

A->accept(drawer)

drawer.visit(A)

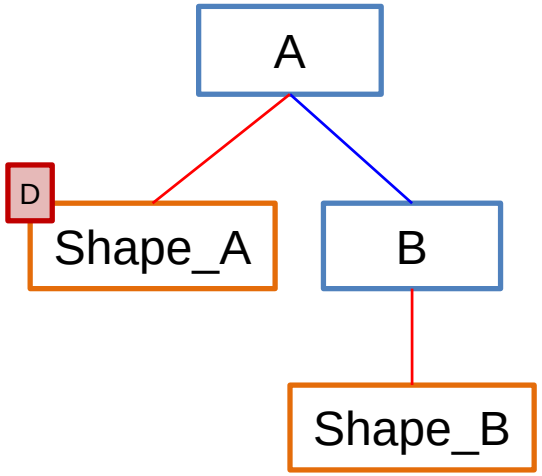
pass ‘drawer’ to Shape_A

drawer.visit(Shape_A)

Operation

Stack = {A}

draw{A * Shape_A}



drawer’s visit() function

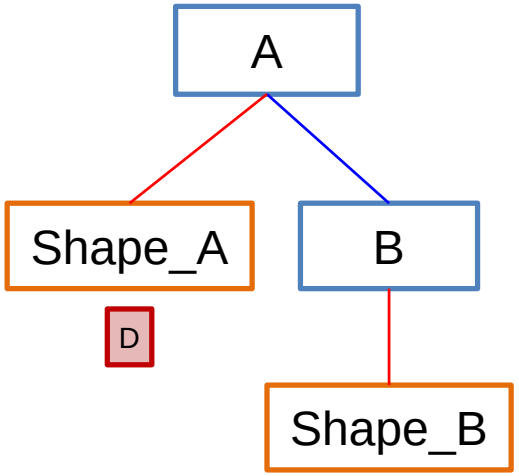
```
virtual bool visit(SgTransformNode& node) {
  rbtStack_.push_back(rbtStack_.back( ) * node.getRbt( ));
  return true;
}

virtual bool visit(SgShapeNode& shapeNode) {
  const Matrix4 MVM = rigTFormToMatrix(rbtStack_.back( )) * shapeNode.getAffineMatrix( );
  sendModelViewNormalMatrix(curSS_, MVM, normalMatrix(MVM));
  shapeNode.draw(curSS_);
  return true;
}
```

accept( ) and visit( ) for drawer

KAIST

Example	Operation
A->accept(drawer)	
drawer.visit(A)	Stack = {A}
pass ‘drawer’ to Shape_A	
drawer.visit(Shape_A)	draw{A * Shape_A}
pass ‘drawer’ to no child	None, Stack = {A}



drawer’s visit() function

```
virtual bool visit(SgTransformNode& node) {
  rbtStack_.push_back(rbtStack_.back( ) * node.getRbt( ));
  return true;
}

virtual bool visit(SgShapeNode& shapeNode) {
  const Matrix4 MVM = rigTFormToMatrix(rbtStack_.back( )) * shapeNode.getAffineMatrix( );
  sendModelViewNormalMatrix(curSS_, MVM, normalMatrix(MVM));
  shapeNode.draw(curSS_);
  return true;
}
```

accept( ) and visit( ) for drawer

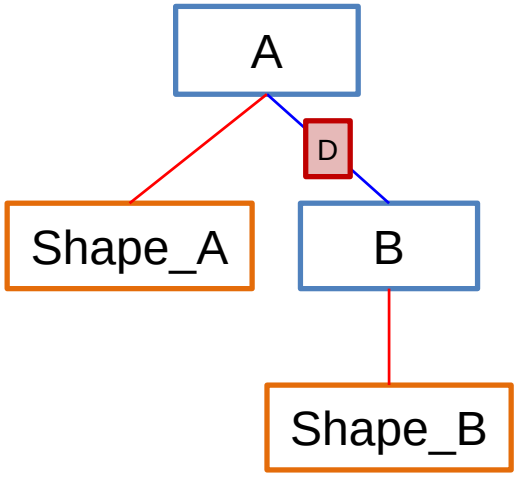
KAIST

Example

pass ‘drawer’ to B

Operation

Stack = {A}



drawer’s visit() function

```
virtual bool visit(SgTransformNode& node) {
  rbtStack_.push_back(rbtStack_.back( ) * node.getRbt( ));
  return true;
}

virtual bool visit(SgShapeNode& shapeNode) {
  const Matrix4 MVM = rigTFormToMatrix(rbtStack_.back( )) * shapeNode.getAffineMatrix( );
  sendModelViewNormalMatrix(curSS_, MVM, normalMatrix(MVM));
  shapeNode.draw(curSS_);
  return true;
}
```

accept( ) and visit( ) for drawer

KAIST

Example

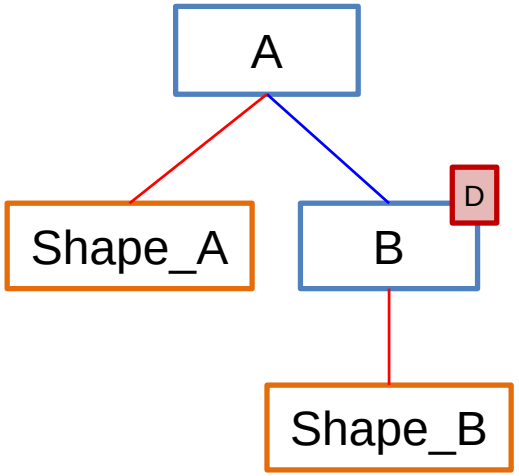
pass ‘drawer’ to B

drawer.visit(B)

Operation

Stack = {A}

Stack = {AB}



drawer’s visit() function

```
virtual bool visit(SgTransformNode& node) {
  rbtStack_.push_back(rbtStack_.back( ) * node.getRbt( ));
  return true;
}

virtual bool visit(SgShapeNode& shapeNode) {
  const Matrix4 MVM = rigTFormToMatrix(rbtStack_.back( )) * shapeNode.getAffineMatrix( );
  sendModelViewNormalMatrix(curSS_, MVM, normalMatrix(MVM));
  shapeNode.draw(curSS_);
  return true;
}
```

accept( ) and visit( ) for drawer

KAIST

Example

Operation

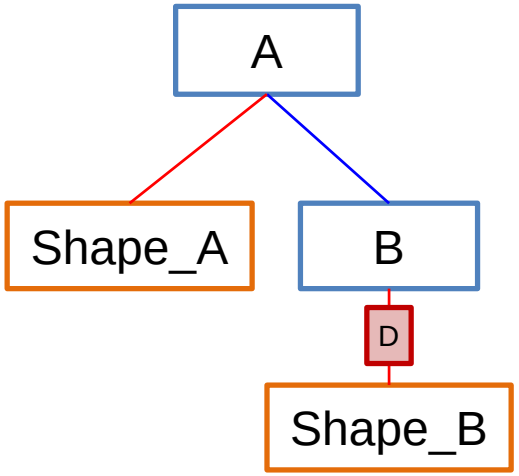
pass ‘drawer’ to B

Stack = {A}

drawer.visit(B)

Stack = {AB}

pass ‘drawer’ to Shape_B



drawer’s visit() function

```
virtual bool visit(SgTransformNode& node) {
  rbtStack_.push_back(rbtStack_.back( ) * node.getRbt( ));
  return true;
}

virtual bool visit(SgShapeNode& shapeNode) {
  const Matrix4 MVM = rigTFormToMatrix(rbtStack_.back( )) * shapeNode.getAffineMatrix( );
  sendModelViewNormalMatrix(curSS_, MVM, normalMatrix(MVM));
  shapeNode.draw(curSS_);
  return true;
}
```

accept( ) and visit( ) for drawer

KAIST

Example

Operation

pass ‘drawer’ to B

Stack = {A}

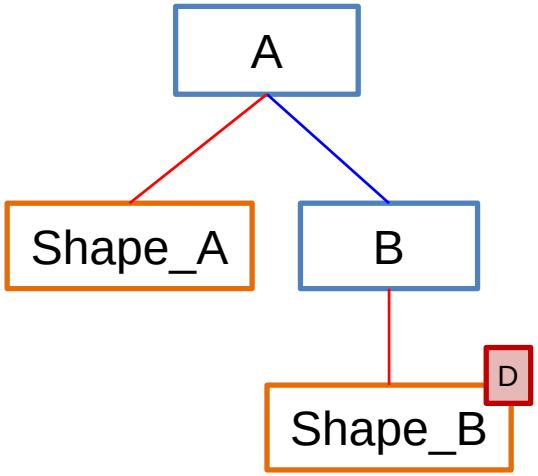
drawer.visit(B)

Stack = {AB}

pass ‘drawer’ to Shape_B

drawer.visit(Shape_B)

draw{AB * Shape_B}



drawer’s visit() function

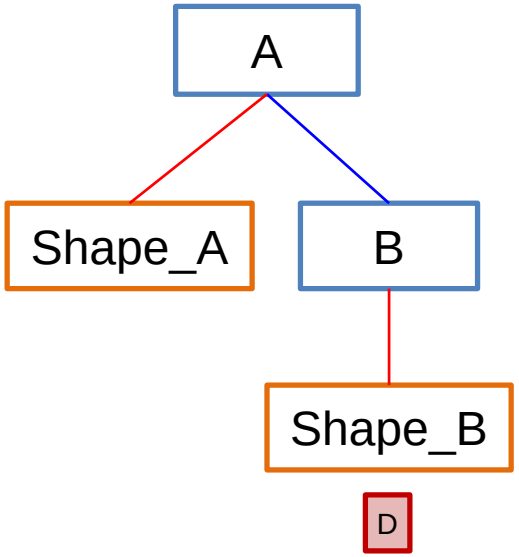
```
virtual bool visit(SgTransformNode& node) {
  rbtStack_.push_back(rbtStack_.back( ) * node.getRbt( ));
  return true;
}

virtual bool visit(SgShapeNode& shapeNode) {
  const Matrix4 MVM = rigTFormToMatrix(rbtStack_.back( )) * shapeNode.getAffineMatrix( );
  sendModelViewNormalMatrix(curSS_, MVM, normalMatrix(MVM));
  shapeNode.draw(curSS_);
  return true;
}
```

accept( ) and visit( ) for drawer

KAIST

Example	Operation
pass ‘drawer’ to B	Stack = {A}
drawer.visit(B)	Stack = {AB}
pass ‘drawer’ to Shape_B	
drawer.visit(Shape_B)	draw{AB * Shape_B}
pass ‘drawer’ to no child	



drawer’s visit() function

```
virtual bool visit(SgTransformNode& node) {
  rbtStack_.push_back(rbtStack_.back( ) * node.getRbt( ));
  return true;
}

virtual bool visit(SgShapeNode& shapeNode) {
  const Matrix4 MVM = rigTFormToMatrix(rbtStack_.back( )) * shapeNode.getAffineMatrix( );
  sendModelViewNormalMatrix(curSS_, MVM, normalMatrix(MVM));
  shapeNode.draw(curSS_);
  return true;
}
```

Task 2. Part Picking

KAIST

- If picking mode is on,
  - do not swap front buffer and back buffer
  - render the different scene on back buffer using colors defined by each object's ID.
  - Handle the h_uIdColor in shaderState using object's ID color.
  - You can query the drawer for the current ShaderState by calling its getCurSS()
  - do not shade according to light direction

Task 2. Part Picking

KAIST

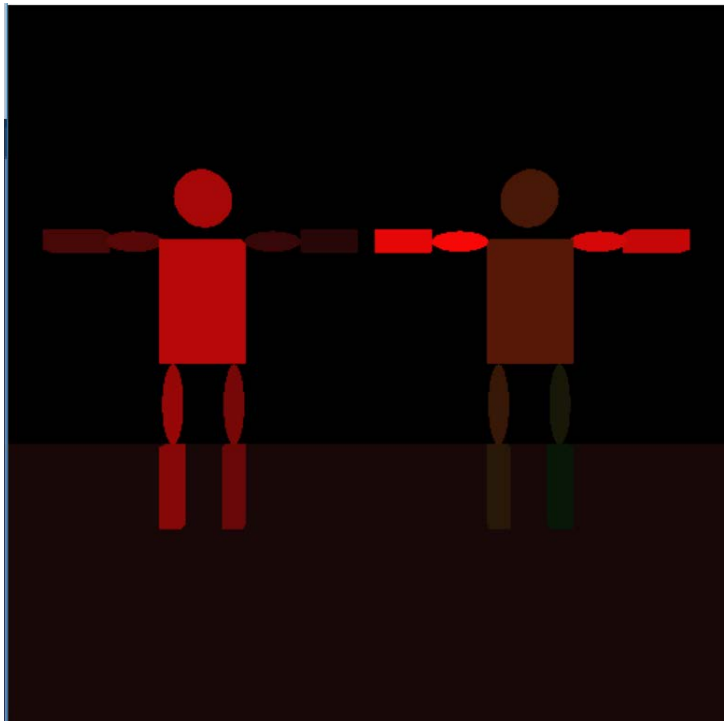
- If some object is picked,
  - Using shared_ptr<SgRbtNode> q = dynamic_pointer_cast<SgRbtNode>(p);
  - SgShapeNode or SgRootNode cannot be cast to SgRbtNode.
  - you can distinguish which object is picked using back buffer's color
  - Using glReadPixels to get RGB value to find object's ID.
  - redisplay(swap) the scene and new arc ball

- Fill TODO in the Picker class

Task 2. Part Picking

KAIST

- You can see the Picking screen to comment glutPostRedisplay() in mouse() and uncomment glutSwapBuffer() in pick()



Task 3. Transform Any Part

KAIST

- Transform any part with keeping hierarchical structure using scene graph

- Refer to RbtAccumVisitor class

Task 4. Build the robot

KAIST

- Now, build your own robot using constructRobot().