Tutorial 4

In tutorial 3 we have learned to rotate an object, and described glPushMatrix() and glPopMatrix() in brief, but their use were not prevalent. In this tutorial we will use these functions to build a hierarchical model. This tutorial is divided in three files viz. HNode.hpp, HNode.cpp and tutorial_04.cpp.

HNode

HNode class is defined to be used for encapsulating any object part and its relation with other parts. HNode class is divided into two files, where HNode.hpp file gives the definition of HNode class while HNode.cpp file has its implementation. Lets look at its definition,

HNode class has the following member variables:

```
class HNode {
private:
  float tx,ty,tz;
  float rx,ry,rz;
  int num_vertices;
  float **vertex_pos;
  float **vertex_col;

  std::vector<HNode*> children;
  HNode* parent;
  ...
};
```

Where,

- tx, ty and tz - defines the translation of a part from its origin in x, y and z axes respectively.
- rx, ry and rz - defines the rotation of a part in x, y and z axes respectively, at the translated point.

- num_vertices - number of vertices in array pointed by vertex_pos and vertex_col
- vertex_pos - pointer to 2D position array of vertices
- vertex_col - pointer to 2D colour array of vertices
- children - a vector of pointers to it's children nodes in hierarchy.
- parent - a pointer to it's parent node in hierarchy.

Following are the member functions of HNode class :

```
class HNode {
...
public:
  //constructor
  HNode (HNode* parent, int num_vertices, float pos_v4[][4],
        float col_v4[][4]);
  //add's a child to this node
  void add_child(HNode*);
  //Drawing function
  void render();
  //changes the current configuration of the node
  void change_parameters(float tx, float ty, float tz,
                         float rx, float ry, float rz);
  //Render itself and it's subtree (all child and their childern's...)
  void render_tree();
  //increment/decrement rotation angles
  void inc_rx();
  void inc_ry();
  void inc_rz();
  void dec_rx();
  void dec_ry();
  void dec_rz();
};
```

The most important function of all these is render_tree. First we do glPushMatrix() so no operation after this and before glPopMatrix() affects the world. Then we render the current node, which affects the current matrix, then we render all it's children, and their children recursively. So any

operation performed on parent also affects its subtree.
The syntax for render_tree function is as follows:
void render_tree(void)

```
void HNode::render_tree(){
  glPushMatrix();
  //Renders itself
  render();
  //Renders subtree below it
  for(int i=0;i<children.size();i++){
    children[i]->render_tree();
  }
  glPopMatrix();
}
```

The syntax for HNode constructor is as follows:
HNode(HNode* par, int num_v, float pos_v4[][4], float col_v4[][4])
Parameters:

- par - pointer to parent node else NULL if no parent.
- num_v - number of vertices in pos_v4 and pos_c4.
- pos_v4 - 2D array of vertex positions.
- col_v4 - 2D array of vertex colours.

```
HNode::HNode(HNode* par, int num_v, float pos_v4[][4],
float col_v4[][4]){
  //Set the parent and add this node as child of it's parent
  if(par!=NULL){
    parent=par;
    parent->add_child(this);
  }

  //Allocate memory for storing copy of positions and colors
  num_vertices = num_v;
  vertex_pos = new float*[num_vertices];
  vertex_col = new float*[num_vertices];
```

```
  //Copy vertices
  for(int i=0;i<num_v;i++){
    vertex_pos[i]=new(float[4]);
    vertexcopy(pos_v4[i],vertex_pos[i]);

    vertex_col[i]=new(float[4]);
    vertexcopy(col_v4[i],vertex_col[i]);
  }

  //Translation and rotation defaults to 0
  tx=ty=tz=rx=ry=rz=0;
}
```

The syntax for add_child function is as follows:
void add_child(HNode *child)
Parameters:

- child - pointer to child node.

```
void HNode::add_child(HNode *child){
  //append the child to node's children list
  children.push_back(child);
}
```

The syntax for render function is as follows:
void render(void)

```
void HNode::render(){
  //Translate the origin
  glTranslatef(tx,ty,tz);
  //Rotate at translated origin
  glRotatef(rx, 1.0, 0.0, 0.0);
  glRotatef(ry, 0.0, 1.0, 0.0);
  glRotatef(rz, 0.0, 0.0, 1.0);

  //Draw triangles
  glBegin(GL_TRIANGLES);
  for (int i=0;i<num_vertices;i++){
    glColor3fv(vertex_col[i]);
    glVertex3fv(vertex_pos[i]);
```

```
  }
  glEnd();
}
```

The syntax for change_parameters function is as follows:
void change_parameters(float tx, float ty, float tz, float rx, float ry, float rz);
Parameters:

- tx, ty and tz - translation form origin over x, y and z axes
- rx, ry and rz - rotation angles over x, y and z axes

```
void HNode::change_parameters(float tx, float ty, float tz,
                              float rx, float ry, float rz){
  //update translation
  this->tx=tx;
  this->ty=ty;
  this->tz=tz;

  //update rotation angles
  this->rx=rx;
  this->ry=ry;
  this->rz=rz;
}
```

The syntax for inc_rx function is as follows:
void inc_rx(void)

```
void HNode::inc_rx(){
  //Increments the rotation angle over x-axis by 1 degree
  rx++;
  //Reset the angle to be in range 0 to 360 degree
  if(rx>360)
  rx-=360;
}
```

Other functions inc_ry and inc_rz are similar.

The syntax for dec_rx function is as follows:
void dec_rx(void)

```
void HNode::dec_rx(){
  //Decrements the rotation angle over x-axis by 1 degree
  rx--;
  //Reset the angle to be in range 0 to 360 degree
  if(rx<360)
  rx+=360;
}
```

Other functions dec_ry and dec_rz are similar.

The syntax for add_child function is as follows:
void add_child(HNode *child)
Parameters:

- child - pointer to child node.

```
void HNode::add_child(HNode *child){
  //append the child to node's children list
  children.push_back(child);
}
```

Coming to tutorial_04.cpp file, it uses the HNode class to implement a three node arm.

```
  float cubiod_corners[8][4]={
    {0,-0.25,-0.25,1},
    {2,-0.25,-0.25,1},
    {2,0.25,-0.25,1},
    {0,0.25,-0.25,1},
    {0,-0.25,0.25,1},
    {0,0.25,0.25,1},
    {2,0.25,0.25,1},
    {2,-0.25,0.25,1}
  };

  makeCuboid(cubiod_corners,v_positions);
```

Defines the eight corners of an arm in homogeneous co-ordinate system and makeCuboid() forms the triangle co-ordinates for cube and saves them in v_positions. Similarly for colours of each corner.

```
  //Parent Node
  node[0] = new HNode(NULL,36,v_positions,v_colors);
```

```
  //Child Nodes
  node[1] = new HNode(node[0],36,v_positions,v_colors);
  node[1]->change_parameters(2.0,0.0,0.0,0.0,0.0,0.0);
  node[2] = new HNode(node[1],36,v_positions,v_colors);
  node[2]->change_parameters(2.0,0.0,0.0,0.0,0.0,0.0);
```

Above code defines the arm configuration, where node[0] is the root node, node[1] is the child of node[0] and is translated by 2 units on x-axis and node[2] is the child of node[1] and is also translated by 2 units on x-axis.

```
void processNormalKeys(unsigned char key, int x, int y) {
  switch(key) {
    case '1':
      curr=0;
      break;
    case '2':
      curr=1;
      break;
    case '3':
      curr=2;
      break;
  }
  if (key == 27)
  exit(0);
}
```

Updated the function processNormalKeys to accommodate control for changing the operational arm, where curr saves the index of current operational node. Pressing the number key 1 makes the root node as current node, pressing the number key 2 makes the second node as current node and pressing the number key 3 makes the last node as current node.