**CS7GV5 Report Example**

|  |  |
| --- | --- |
| **Name:** | Zhiqiang Cheng |
| **Student ID:** | 22322639 |
| **Youtube link:** | https://youtu.be/AzgBT6INkzA |

|  |
| --- |
| **Required feature:** Simple 2-bone IK in 2D |
| *Screenshot(s) of feature:*  形状, 箭头  描述已自动生成  形状, 箭头  描述已自动生成 |
| *Describe your implementation:*  形状, 箭头  描述已自动生成   1. press key up, down, left, right to move target 2. forward kinematics   press z or x, to rotate bone 1  press a or s, to rotate bone 2  press q or w, to rotate bone 3   1. IK with CCD   press c |
| *Code Snippet:*  void display() {  int currentFrame = glutGet(GLUT\_ELAPSED\_TIME);  deltaTime = currentFrame - lastFrame;  lastFrame = currentFrame;  // tell GL to only draw onto a pixel if the shape is closer to the viewer  glEnable(GL\_DEPTH\_TEST); // enable depth-testing  glDepthFunc(GL\_LESS); // depth-testing interprets a smaller value as "closer"  glClearColor(0.5f, 0.5f, 0.5f, 1.0f);  glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);  shader.use();    mat4 view = identity\_mat4();  //mat4 persp\_proj = perspective(45.0f, (float)width / (float)height, 0.1f, 1000.0f);  mat4 persp\_proj = perspective(camera.Zoom, (float)scr\_width / (float)scr\_height, 0.1f, 100.0f);  //view = look\_at(vec3(0, 0, 0), vec3(0, 0, -1), vec3(0, 1, 0));  view = camera.GetViewMatrix();  shader.setMat4("proj", persp\_proj);  shader.setMat4("view", view);  // box, target  modelTarget = identity\_mat4();  modelTarget = scale(modelTarget, vec3(0.2,0.2,0.2));  modelTarget = rotate\_x\_deg(modelTarget, rotate\_x);  modelTarget = rotate\_y\_deg(modelTarget, rotate\_y);  modelTarget = rotate\_z\_deg(modelTarget, rotate\_z);  //modelTarget = translate(modelTarget, vec3(0.0, -3.5, 0.0));  modelTarget = translate(modelTarget, vec3(target\_x, target\_y, target\_z));  shader.setMat4("model", modelTarget);  renderCube();  // bone 1, root  model1 = identity\_mat4();  model1 = rotate\_z\_deg(model1, bone1\_start\_z\_deg + bone1\_rotate\_z);  shader.setMat4("model", model1);  renderBone();  // bone 2  model2 = identity\_mat4();  model2 = rotate\_z\_deg(model2, bone2\_start\_z\_deg + bone2\_rotate\_z);  model2 = translate(model2, vec3(-1.05, 0.0, 0.0));  model2 = model1 \* model2;  shader.setMat4("model", model2);  renderBone();  // bone 3  model3 = identity\_mat4();  model3 = rotate\_z\_deg(model3, bone3\_start\_z\_deg + bone3\_rotate\_z);  model3 = translate(model3, vec3(-1.05, 0.0, 0.0));  model3 = model2 \* model3;  shader.setMat4("model", model3);  renderBone();  // end point  model4 = identity\_mat4();  model4 = scale(model4, vec3(0.5, 0.5, 0.5));  model4 = rotate\_z\_deg(model4, bone4\_start\_z\_deg + bone4\_rotate\_z);  model4 = translate(model4, vec3(-1.05, 0.0, 0.0));  model4 = model3 \* model4;  shader.setMat4("model", model4);  //renderBone();  glutSwapBuffers();  }  void SpecialKeys(int key, int x, int y) {  GLfloat stepSize = 0.025f;  switch (key) {  case GLUT\_KEY\_UP:  target\_y += stepSize;  break;  case GLUT\_KEY\_DOWN:  target\_y -= stepSize;  break;  case GLUT\_KEY\_LEFT:  target\_x -= stepSize;  break;  case GLUT\_KEY\_RIGHT:  target\_x += stepSize;  break;  default:  break;  }  }  void CCD(int frame) {  static int i = 0;  if (++i > 100) {  i = 0;  CCDRunning = false;  return;  }    vec3 endPos = getPos(model4);  vec3 tarPos = getPos(modelTarget);  vec3 curPos;  vec3 e\_c;  vec3 t\_c;  vec3 r;  float cos\_theta;  float angle;  float distance = sqrt(pow((tarPos.v[0] - endPos.v[0]), 2) + pow((tarPos.v[1] - endPos.v[1]), 2));  float tolerance = 0.1;  if (distance < tolerance) {  solve = true;  CCDRunning = false;  }  else  {  if (frame == 1) {  endPos = getPos(model4);  tarPos = getPos(modelTarget);  curPos = getPos(model3);  e\_c = normalise(endPos - curPos);  t\_c = normalise(tarPos - curPos);  r = cross(e\_c, t\_c);  cos\_theta = clamp(dot(e\_c, t\_c));  angle = acos(cos\_theta) \* ONE\_RAD\_IN\_DEG;  if (r.v[2] < 0) {  angle \*= -1;  }  bone3\_rotate\_z += angle;  }  else if (frame == 2) {  endPos = getPos(model4);  tarPos = getPos(modelTarget);  curPos = getPos(model2);  e\_c = normalise(endPos - curPos);  t\_c = normalise(tarPos - curPos);  r = cross(e\_c, t\_c);  cos\_theta = clamp(dot(e\_c, t\_c));  angle = acos(cos\_theta) \* ONE\_RAD\_IN\_DEG;  if (r.v[2] < 0) {  angle \*= -1;  }  bone2\_rotate\_z += angle;  }  else if (frame == 3) {  endPos = getPos(model4);  tarPos = getPos(modelTarget);  curPos = getPos(model1);  e\_c = normalise(endPos - curPos);  t\_c = normalise(tarPos - curPos);  r = cross(e\_c, t\_c);  cos\_theta = clamp(dot(e\_c, t\_c));  angle = acos(cos\_theta) \* ONE\_RAD\_IN\_DEG;  if (r.v[2] < 0) {  angle \*= -1;  }  bone1\_rotate\_z += angle;  }  }  } |

|  |
| --- |
| **Required feature:** Multi-bone IK in 3D |
| *Screenshot(s) of feature:*  图片包含 形状  描述已自动生成  图片包含 雷达图  描述已自动生成 |
| *Describe your implementation:*  图示, 形状, 箭头  描述已自动生成   1. There are total 15 bones，contains five big bone (1,2,3,4,5), 10 small bones (6-15) 2. press key up, down, left, right to move target 3. forward kinematics   press z or x, to rotate bone 1  press a or s, to rotate bone 2  press q or w, to rotate bone 3  press d or f, to rotate bone 4  press e or r, to rotate bone 5   1. IK with CCD   press c |
| *Code Snippet:*  void display() {  int currentFrame = glutGet(GLUT\_ELAPSED\_TIME);  deltaTime = currentFrame - lastFrame;  lastFrame = currentFrame;  // tell GL to only draw onto a pixel if the shape is closer to the viewer  glEnable(GL\_DEPTH\_TEST); // enable depth-testing  glDepthFunc(GL\_LESS); // depth-testing interprets a smaller value as "closer"  glClearColor(0.5f, 0.5f, 0.5f, 1.0f);  glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);  shader.use();    mat4 view = identity\_mat4();  //mat4 persp\_proj = perspective(45.0f, (float)width / (float)height, 0.1f, 1000.0f);  mat4 persp\_proj = perspective(camera.Zoom, (float)scr\_width / (float)scr\_height, 0.1f, 100.0f);  //view = look\_at(vec3(0, 0, 0), vec3(0, 0, -1), vec3(0, 1, 0));  view = camera.GetViewMatrix();  shader.setMat4("proj", persp\_proj);  shader.setMat4("view", view);  // box, target  modelTarget = identity\_mat4();  modelTarget = scale(modelTarget, vec3(0.2,0.2,0.2));  modelTarget = rotate\_x\_deg(modelTarget, rotate\_x);  modelTarget = rotate\_y\_deg(modelTarget, rotate\_y);  modelTarget = rotate\_z\_deg(modelTarget, rotate\_z);  //modelTarget = translate(modelTarget, vec3(0.0, -3.5, 0.0));  modelTarget = translate(modelTarget, vec3(target\_x, target\_y, target\_z));  shader.setMat4("model", modelTarget);  renderCube();  // bone 1, root  model[0] = identity\_mat4();  model[0] = rotate\_z\_deg(model[0], 0 + bone\_rotate\_z[0]);  //model1 = translate(model1, vec3(0.0, 0.0, 0.0));  shader.setMat4("model", model[0]);  renderBone();  for (int i = 1; i < numsOfBones; i++)  {  model[i] = identity\_mat4();  model[i] = rotate\_z\_deg(model[i], 0 + bone\_rotate\_z[i]);  model[i] = translate(model[i], vec3(-1.05, 0.0, 0.0));  model[i] = model[i-1] \* model[i];  shader.setMat4("model", model[i]);  renderBone();  // bone left  mat4 model\_left = identity\_mat4();  model\_left = scale(model\_left, vec3(0.5, 0.5, 0.5));  model\_left = rotate\_z\_deg(model\_left, 90 + bone\_left\_rotate\_z);  model\_left = translate(model\_left, vec3(-1.05, 0.0, 0.0));  model\_left = model[i-1] \* model\_left;  shader.setMat4("model", model\_left);  renderBone();  // bone right  mat4 model\_right = identity\_mat4();  model\_right = scale(model\_right, vec3(0.5, 0.5, 0.5));  model\_right = rotate\_z\_deg(model\_right, -90 + bone\_right\_rotate\_z);  model\_right = translate(model\_right, vec3(-1.05, 0.0, 0.0));  model\_right = model[i-1] \* model\_right;  shader.setMat4("model", model\_right);  renderBone();  }    // top1  modelTop1 = identity\_mat4();  modelTop1 = scale(modelTop1, vec3(0.5, 0.5, 0.5));  modelTop1 = rotate\_z\_deg(modelTop1, boneTop1\_start\_z\_deg + boneTop1\_rotate\_z);  modelTop1 = translate(modelTop1, vec3(-1.05, 0.0, 0.0));  modelTop1 = model[numsOfBones-1] \* modelTop1;  shader.setMat4("model", modelTop1);  renderBone();  // top2  modelTop2 = identity\_mat4();  modelTop2 = scale(modelTop2, vec3(0.5, 0.5, 0.5));  modelTop2 = rotate\_z\_deg(modelTop2, boneTop2\_start\_z\_deg + boneTop2\_rotate\_z);  modelTop2 = translate(modelTop2, vec3(-1.05, 0.0, 0.0));  modelTop2 = model[numsOfBones-1] \* modelTop2;    shader.setMat4("model", modelTop2);  renderBone();  glutSwapBuffers();  }  void CCD(int frame) {  static int i = 0;  if (++i > numsOfBones \* 30) {  i = 0;  CCDRunning = false;  return;  }    vec3 endPos = getPos(modelTop1);  vec3 tarPos = getPos(modelTarget);  vec3 curPos;  vec3 e\_c;  vec3 t\_c;  vec3 r;  float cos\_theta;  float angle;  float distance = sqrt(pow((tarPos.v[0] - endPos.v[0]), 2) + pow((tarPos.v[1] - endPos.v[1]), 2));  float tolerance = 0.1;  if (distance < tolerance) {  solve = true;  CCDRunning = false;  }  else  {    int frameIndex = numsOfBones - frame - 1;  for (int i = numsOfBones - 1; i >= 0; i--)  {  if (frameIndex == i) {  endPos = getPos(modelTop1);  tarPos = getPos(modelTarget);  curPos = getPos(model[i]);  e\_c = normalise(endPos - curPos);  t\_c = normalise(tarPos - curPos);  r = cross(e\_c, t\_c);  cos\_theta = clamp(dot(e\_c, t\_c));  angle = acos(cos\_theta) \* ONE\_RAD\_IN\_DEG;  if (r.v[2] < 0) {  angle \*= -1;  }  bone\_rotate\_z[i] += angle;  }  }  }  } |

|  |
| --- |
| **Required feature:** Scripted animation |
| *Screenshot(s) of feature:*  图片包含 形状  描述已自动生成  图片包含 形状  描述已自动生成 |
| *Describe your implementation:*   1. The small bone（6 – 15），can animated at 45 degree. 2. The big bone（1，2，3，4，5），can animated at 360 fegree.   press z or x, to animate bone 1  press a or s, to animate bone 2  press q or w, to animate bone 3  press d or f, to animate bone 4  press e or r, to animate bone 5   1. IK with CCD   press c to animate |
| *Code Snippet:*  void updateScene() {  static DWORD last\_time = 0;  DWORD curr\_time = timeGetTime();  if (last\_time == 0)  last\_time = curr\_time;  float delta = (curr\_time - last\_time) \* 0.001f;  last\_time = curr\_time;  // animte the target  rotate\_y += 20.0f \* delta;  rotate\_x += 20.0f \* delta;  rotate\_z += 20.0f \* delta;  // animte the bone  boneTop1\_rotate\_z += 20.0f \* delta;  boneTop1\_rotate\_z = fmodf(boneTop1\_rotate\_z, 45.0f);  boneTop2\_rotate\_z -= 20.0f \* delta;  boneTop2\_rotate\_z = fmodf(boneTop2\_rotate\_z, -45.0f);  // animte the bone  bone\_left\_rotate\_z += 70.0f \* delta;  bone\_left\_rotate\_z = fmodf(bone\_left\_rotate\_z, 90.0f);  bone\_right\_rotate\_z -= 70.0f \* delta;  bone\_right\_rotate\_z = fmodf(bone\_right\_rotate\_z, -90.0f);  // CCD and animation  if (CCDRunning) {  static int frame = 0;  CCD(frame);  frame++;  if (frame > numsOfBones) frame = 0;  }  // Draw the next frame  glutPostRedisplay();  } |