

University of Dublin



TRINITY COLLEGE

**Animation Inverse kinematic**

Student Name: Hao Guan  
Student Number: 14305641  
Submission Date: 13/11/2014  
Course Module: Real-time Animation

School of Computer Science and Statistics

O'Reilly Institute, Trinity College, Dublin 2, Ireland

## Summary

The goal of this Inverse kinematic lab is to develop a human full arm and hand with bone structure to simulate Inverse kinematic base on the CCD algorithm, and few feature was added as extra, such as joint limitation, hand grabbing, manual control and cubic spline interpolation with keyframe.

## Code explanation

Bone class:

```
10 class Bone
11 {
12 public:
13     Bone(void);
14     ~Bone(void);
15
16     int id;
17     glm::vec3 pos;
18     glm::mat4 offset;
19     glm::mat4 localTransformation;
20     glm::mat4 globalTransformation;
21     Bone* children[1];
22     Bone* parent;
23     float damp_width;
24     int childrenSize;
25     float max_rx;
26     float min_rx;
27     float max_ry;
28     float min_ry;
29     float max_rz;
30     float min_rz;
31     float currentXPos;
32     float currentYPos;
33     float currentZPos;
34
35     Bone* createBone(int ID, glm::vec3 pos, float damp_width);
36     void addChild(Bone* child);
37     void addParent(Bone* parent);
38     int getID(Bone* bone);
39     glm::vec3 getPos(Bone* bone);
40     void setJointLimit(float max_rx, float min_rx, float max_ry, float min_ry, float max_rz, float min_rz);
41 };
```

For this lab, the new maximum rotation on each axis is added, also the current bone rotation degree of each axis, this will allow calculate the each joint limitation.

```
45 void Bone::setJointLimit(float max_rx, float min_rx, float max_ry, float min_ry, float max_rz, float min_rz)
46 {
47     this->max_rx = max_rx;
48     this->min_rx = min_rx;
49     this->max_ry = max_ry;
50     this->min_ry = min_ry;
51     this->max_rz = max_rz;
52     this->min_rz = min_rz;
53 }
```

```
39
40     handNode[0]->setJointLimit(10.0f,-179.0f,90.0f,-10.0f,45.0f,-179.0f);
41     handNode[1]->setJointLimit(135.0f,0.0f,90.0f,0.0f,0.0f,0.0f);
42     handNode[2]->setJointLimit(75.0f,-75.0f,0.0f,0.0f,45.0f,-45.0f);
43
```

Joint limitation for upper arm, lower arm and wrist.

## Skeleton class:

```
13 class ArmSkeleton
14 {
15 public:
16     ArmSkeleton(Setup* m_setup);
17     ~ArmSkeleton(void);
18
19     Bone* bone;
20     Bone* handNode[19];
21     Cylinder* cylinder[19];
22     Cylinder* armTarget;
23     Setup* m_setup;
24
25     double lastTime;
26     double currentTime;
27     float deltaTime;
28     bool timeFlag;
29     float whatsoever;
30     float speed;
31     glm::vec3 armTargetPos;
32     glm::mat4 armTargetTransformation;
33     float joint2TargetPos;
34     float endToTargeDistance;
35     float currentXPos,currentYPos,currentZPos;
36     float DOF_x,DOF_y,DOF_z;
37     float deltaT;
38     bool targetLoops;
39
40
41     void createArmNode();
42     void drawArmMesh(GLuint shaderProgramID);
43     void updateArmMesh(GLuint shaderProgramID);
44     void updateArmTarget(GLuint shaderProgramID);
45     void calculateInverseKinematics();
46     void calcGlobalTransformation();
47     glm::quat calcJointLimit(Bone* bone, glm::vec3 angles);
48     void calcEffectorToTargetDistance();
49     glm::vec3 interpolateCubic(float deltaTime, glm::vec3 beingPos, glm::vec3 point1, glm::vec3 point2, glm::vec3 endPos);
50 };
```

In skeleton class I have implement few more new functions, such as calculate IK with CCD, calculate joint limitation, calculate the effector to target distance and cubic spline interpolation.

The code for calculate IK with CCD steps as follow:

Get the position from effector position, current bone position and target position. From this 3 position we can get the cos angel from dot product, after that we will get the rotation from cross product, once we get both of them, we can rotation quaternion and then transfer to matrix to apply to bone local transformation.

## Calculate IK with CCD

```
355 void ArmSkeleton::calculateInverseKinematics()
356 {
357     int NumConter,linker;
358
359     NumConter = 0;
360     linker = handNode[2]->id;
361     while (endToTargetDistance > IK_POS_THRESH && NumConter < MAX_IK_TRIES)
362     {
363         glm::vec3 effectorPos = glm::vec3(handNode[3]->globalTransformation[3][0],
364             handNode[3]->globalTransformation[3][1], handNode[3]->globalTransformation[3][2]);
365         glm::vec3 bonePos = glm::vec3(handNode[linker]->globalTransformation[3][0],
366             handNode[linker]->globalTransformation[3][1], handNode[linker]->globalTransformation[3][2]);
367
368         glm::vec3 endVector = effectorPos - bonePos;
369         glm::vec3 endVectorNor = glm::normalize(endVector);
370
371         glm::vec3 targetVector = armTargetPos - bonePos;
372         glm::vec3 targetVectorNor = glm::normalize(targetVector);
373
374         float cosAngle = glm::dot(targetVectorNor, endVectorNor);
375
376         if ((endVectorNor != endVectorNor)|| (targetVectorNor != targetVectorNor))
377         {
378             break;
379         }
380
381         if (cosAngle >= 1)
382         {
383             break;
384         }
385
386         if (cosAngle <= -1)
387         {
388             break;
389         }
390
391         glm::vec3 crossResult = glm::cross(endVectorNor, targetVectorNor);
392         crossResult = glm::normalize(crossResult);
393         crossResult = glm::vec3(glm::mat3(glm::inverse(
394             handNode[linker]->globalTransformation)) * crossResult);
395
396         float turnAngle = glm::acos(cosAngle); // GET THE ANGLE
397         float turnDeg = glm::degrees(turnAngle); // COVERT TO DEGREES
398
399         glm::quat eulerRot = glm::angleAxis(turnDeg, crossResult);
400         glm::vec3 angles = glm::eulerAngles(eulerRot);
401         eulerRot = calcJointLimit(handNode[linker], angles);
402         glm::mat4 rot = glm::toMat4(eulerRot);
403
404         handNode[linker]->localTransformation *= rot;
405
406         calcGlobalTransformation();
407         calcEffectorToTargetDistance();
408
409         NumConter++;
410         linker--;
411         if (linker < 0)
412         {
413             linker = handNode[2]->id;
414         }
415     }
416     calcEffectorToTargetDistance();
417 }
```

## Calculate joint limitation

```
400     glm::vec3 angles = glm::eulerAngles(eulerRot);
401     eulerRot = calcJointLimit(handNode[linker], angles);

420     glm::quat ArmSkeleton::calcJointLimit(Bone* bone, glm::vec3 angles)
421     {
422         bool fx = false;
423         bool fy = false;
424         bool fz = false;
425         if (angles.x > 0 && bone->currentXPos == bone->max_rx)
426         {
427             angles.x = 0;
428             fx = true;
429         }
430         if (angles.y > 0 && bone->currentYPos == bone->max_ry)
431         {
432             angles.y = 0;
433             fy = true;
434         }
435         if (angles.z > 0 && bone->currentZPos == bone->max_rz)
436         {
437             angles.z = 0;
438             fz = true;
439         }
440
441         if (angles.x < 0 && bone->currentXPos == bone->min_rx)
442         {
443             angles.x = 0;
444             fx = true;
445         }
446         if (angles.y < 0 && bone->currentYPos == bone->min_ry)
447         {
448             angles.y = 0;
449             fy = true;
450         }
451         if (angles.z < 0 && bone->currentZPos == bone->min_rz)
452         {
453             angles.z = 0;
454             fz = true;
455         }
456     }
```

If the current angles equals maximum rotation axis, 0 degree will be return, as this no rotation allow in this case.

```

450
457     if((angles.x + bone->currentXPos) <= bone->max_rx && (angles.x + bone->currentXPos) >= bone->min_rx && fx == false)
458     {
459         bone->currentXPos = bone->currentXPos + angles.x;
460     }
461     else
462     {
463         bool swicher = false;
464         if ((angles.x + bone->currentXPos) > bone->max_rx)
465         {
466             angles.x = bone->max_rx - bone->currentXPos;
467             bone->currentXPos = bone->max_rx;
468             swicher = true;
469         }
470
471         if ((angles.x + bone->currentXPos) < bone->min_rx && swicher == false)
472         {
473             angles.x = bone->min_rx - bone->currentXPos;
474             bone->currentXPos = bone->min_rx;
475         }
476     }
477

```

If the angle plus current angle is in the limitation range, current position will replaced with new degree, and the angle for rotation is given a new suitable value base on the case of over the maximum of less than minimum degree.

```

478     if((angles.y + bone->currentYPos) <= bone->max_ry && (angles.y + bone->currentYPos) >= bone->min_ry && fy == false)
479     {
480         bone->currentYPos = bone->currentYPos + angles.y;
481     }
482     else
483     {
484         bool swicher = false;
485         if ((angles.y + bone->currentYPos) > bone->max_ry)
486         {
487             angles.y = bone->max_ry - bone->currentYPos;
488             bone->currentYPos = bone->max_ry;
489             swicher = true;
490         }
491
492         if ((angles.y + bone->currentYPos) < bone->min_ry && swicher == false)
493         {
494             angles.y = bone->min_ry - bone->currentYPos;
495             bone->currentYPos = bone->min_ry;
496         }
497     }
498
499
500     if((angles.z + bone->currentZPos) <= bone->max_rz && (angles.z + bone->currentZPos) >= bone->min_rz && fz == false)
501     {
502         bone->currentZPos = bone->currentZPos + angles.z;
503     }
504     else
505     {
506         bool swicher = false;
507         if ((angles.z + bone->currentZPos) > bone->max_rz)
508         {
509             angles.z = bone->max_rz - bone->currentZPos;
510             bone->currentZPos = bone->max_rz;
511             swicher = true;
512         }
513
514         if ((angles.z + bone->currentZPos) < bone->min_rz && swicher == false)
515         {
516             angles.z = bone->min_rz - bone->currentZPos;
517             bone->currentZPos = bone->min_rz;
518         }
519     }
520
521     angles = glm::radians(angles);
522
523     return glm::quat(angles);
524 }
525

```

And do the same for y and x axis, after the new quaternion returned.

## Calculate the effector to target distance

```
527 void ArmSkeleton::calcEffectorToTargetDistance()
528 {
529     endToTargeDistance = glm::distance(glm::vec3(handNode[3]->globalTransformation[3][0],
530     handNode[3]->globalTransformation[3][1], handNode[3]->globalTransformation[3][2]), armTargetPos);
531 }
```

Calculate hand grabbing based on effector to target distance, also update the global transformation after that.

```
311 void ArmSkeleton::calcGlobalTransformation()
312 {
313     if (endToTargeDistance < 2.0f)
314     {
315         for (int i = 4; i < 19; i++)
316         {
317             handNode[i]->localTransformation = glm::rotate(handNode[i]->offset, deltaTime, glm::vec3(1,0,0));
318         }
319         deltaTime += 0.1f;
320         if (deltaTime > 40.0f)
321         {
322             deltaTime = 30.0f;
323         }
324     }
325
326     if (endToTargeDistance >= 2.0f)
327     {
328         for (int i = 4; i < 19; i++)
329         {
330             handNode[i]->localTransformation = glm::rotate(handNode[i]->offset, deltaTime, glm::vec3(1,0,0));
331         }
332         deltaTime -= 0.1f;
333         if (deltaTime < 0.0f)
334         {
335             deltaTime = 0.0f;
336         }
337     }
338
339     handNode[16]->localTransformation = glm::rotate(handNode[16]->offset, 25.0f, glm::vec3(0,0,-1));
340
341     for (int i = 0 ; i < HAND_NODE_NUM ; i++)
342     {
343         if (handNode[i]->id == 0)
344         {
345             handNode[i]->globalTransformation = handNode[i]->localTransformation;
346         }
347         else
348         {
349             handNode[i]->globalTransformation = handNode[i]->parent->globalTransformation * handNode[i]->localTransformation;
350         }
351     }
352 }
```

## Cubic spline interpolation with keyframe

```
294 glm::vec3 ArmSkeleton::interpolateCubic(float deltaTime, glm::vec3 beingPos, glm::vec3 point1,
295                                       glm::vec3 point2, glm::vec3 endPos)
296 {
297     glm::vec3 p1,p2,p3,p4;
298
299     p1 = point2 - endPos - point1 + beingPos;
300     p2 = point1 - beingPos - p1;
301     p3 = endPos - point1;
302     p4 = beingPos;
303
304     return (p1 * pow(deltaTime,3) + p2 * pow(deltaTime,2) + p3 * pow(deltaTime,1) + p4);
305 }
306
307 glm::vec3 ArmSkeleton::interpolateCubic_2(float deltaTime, glm::vec3 beingPos,
308                                           glm::vec3 point1, glm::vec3 point2, glm::vec3 endPos)
309 {
310     return pow(1-deltaTime,3)*beingPos+3*deltaTime*pow(1-deltaTime,2)*point1+3*
311           pow(deltaTime,2)*(1-deltaTime)*point2+pow(deltaTime,3)*endPos;
312 }
```

And the following is when I calling this functions to draw two paths.

```
270     if (targetLoops == false)
271     {
272         armTargetPos = interpolateCubic(deltaT, glm::vec3(20,50,15), glm::vec3(130,0,15),
273                                       glm::vec3(-130,0,15), glm::vec3(-30,10,10));
274
275         if (armTargetPos == glm::vec3(-30,10,10))
276         {
277             targetLoops = true;
278         }
279     }
280
281     if (targetLoops == true)
282     {
283         armTargetPos = interpolateCubic(deltaT, glm::vec3(-30,10,10), glm::vec3(-20,-10,10),
284                                       glm::vec3(15,-20,-10), glm::vec3(20,50,15));
285
286         if (armTargetPos == glm::vec3(20,50,15))
287         {
288             targetLoops = false;
289         }
290     }
```

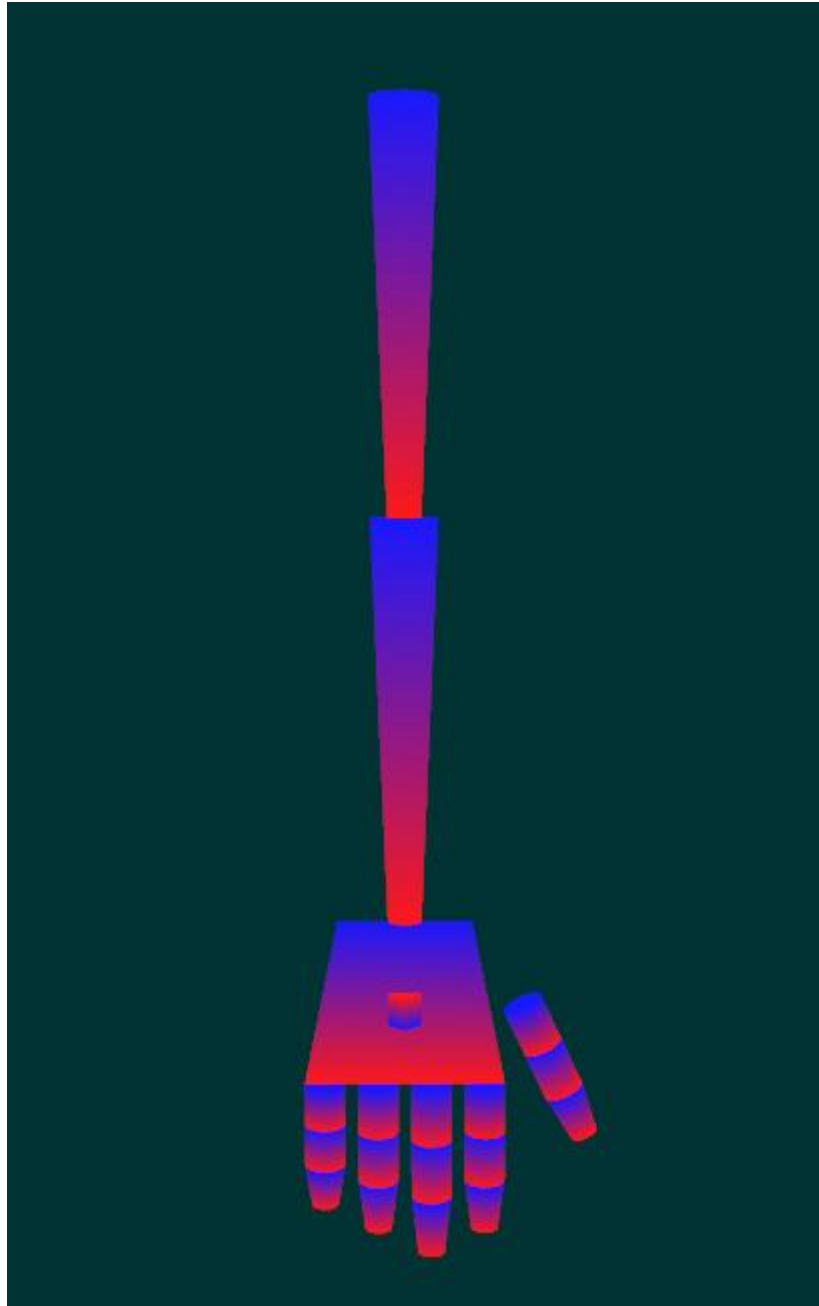


## Manual control for the target

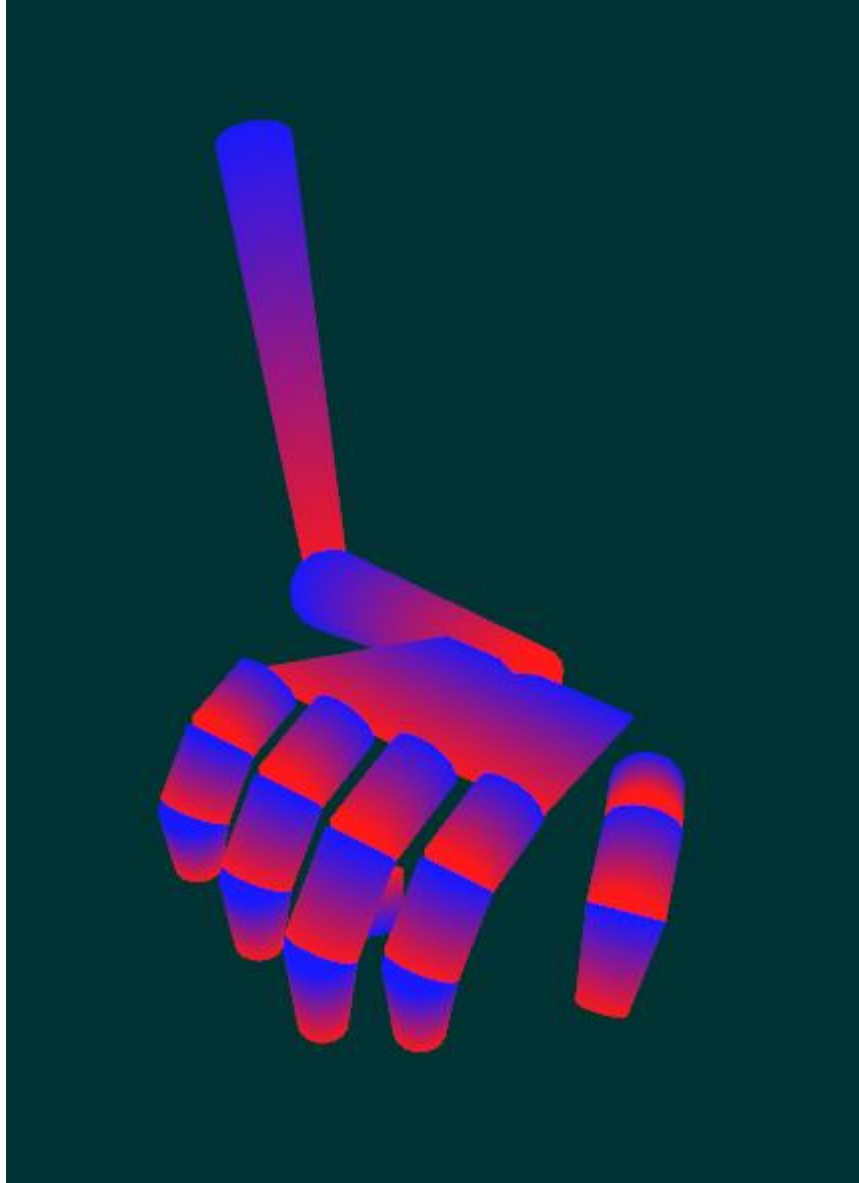
```
232 void ArmSkeleton::updateArmTarget(GLuint shaderProgramID)
233 {
234     //Move y up
235     if (glfwGetKey( m_setup->getWindow(), GLFW_KEY_UP ) == GLFW_PRESS){
236         armTargetPos.y += 0.3;
237     }
238
239     // Move y down
240     if (glfwGetKey( m_setup->getWindow(), GLFW_KEY_DOWN ) == GLFW_PRESS){
241         armTargetPos.y -= 0.3;
242     }
243
244     // Move x right
245     if (glfwGetKey( m_setup->getWindow(), GLFW_KEY_RIGHT ) == GLFW_PRESS){
246         armTargetPos.x += 0.3;
247     }
248
249     // Move x left
250     if (glfwGetKey( m_setup->getWindow(), GLFW_KEY_LEFT ) == GLFW_PRESS){
251         armTargetPos.x -= 0.3;
252     }
253
254     // Move z forward
255     if (glfwGetKey( m_setup->getWindow(), GLFW_KEY_I ) == GLFW_PRESS){
256         armTargetPos.z -= 0.3;
257     }
258
259     // Move z backward
260     if (glfwGetKey( m_setup->getWindow(), GLFW_KEY_K ) == GLFW_PRESS){
261         armTargetPos.z += 0.3;
262     }
```

```
292 armTargetTransformation = glm::translate(glm::mat4(1),glm::vec3(armTargetPos.x,armTargetPos.y-3,armTargetPos.z));
293 armTarget->update(armTargetTransformation, shaderProgramID);
294 armTarget->draw();
```

Hand model



Initial position



Hand grabbing

Hand grabbing in steps.

