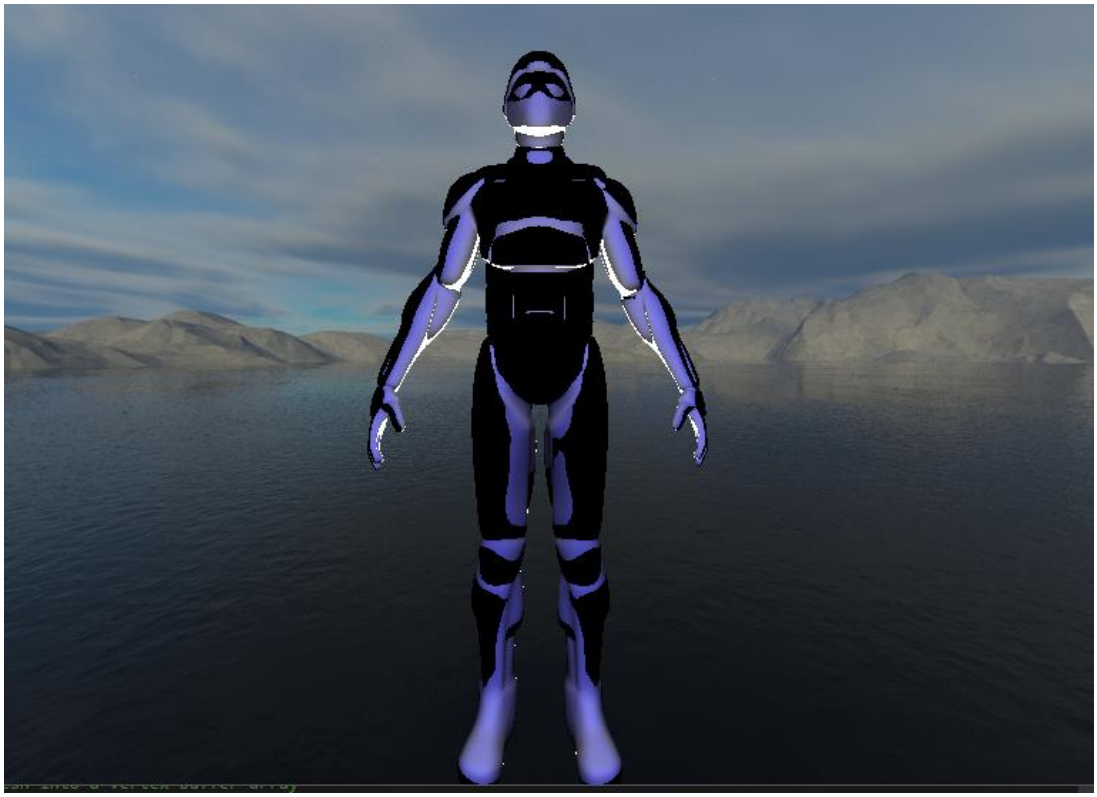
CS7GV5-A-SEM202-201920 REAL-TIME ANIMATION
Assignment 2 – Inverse Kinematics

- **The Humanoid Model:**



- **Analytical Solution for Computing Angles:**

I've used the simple law of cosines and trigonometric functions to create my own function for the analytical solution.

```
//reference: https://appliedgo.net/roboticarm/
float lawOfCosine(float a, float b, float c)
{
    return acos((a * a + b * b - c * c) / (2 * a * b));            //angle = arccos((a^2+b^2-c^2)/2ab)
}

float Angle::calcDist(float x1, float y1, float x2, float y2)
{
    return sqrt(pow(x2 - x1, 2) + pow(y2 - y1, 2));
}

vec2 Angle::calcTheta(vec2 dest)
{
    dist = calcDist(source.v[0], source.v[1], dest.v[0], dest.v[1]);
    theta_t1 = atan2(dest.v[1] - source.v[1], dest.v[0] - source.v[0]);
    theta_t2 = lawOfCosine(dist, l1, l2);
    //std::cout << '\n'<<dist << " " << theta_t1 << " " << theta_t2;
    theta_1 = theta_t1 + theta_t2;                  //upper arm angle

    theta_2 = lawOfCosine(l1, l2, dist);            //lower arm angle

    return vec2(theta_1, theta_2);
}
```

- **Jacobian:**

  Created a function for Jacobian also. Although, I was not able to get the exact joint positions successively after changes which resulted in unsuccessful implementation of the function.

```
//reference: https://medium.com/unity3danimation/overview-of-jacobian-ik-a33939639ab2
void Angle::jacobian(vec3 targetPos)
{
    float eps = 1.0;
    float h = 0.00000001;
    target = targetPos;
    while (abs(calcDist(endPosition.v[0], endPosition.v[0],  target.v[0], target.v[1]) > eps))
    {
        vec3 dO = getDeltaOrientation();
        orient += dO * h;
    }
}

vec3 Angle :: getDeltaOrientation()
{
    mat3 jacob_transpose = getJacobianTranspose();
    vec3 v = target - endPosition;
    vec3 dO = jacob_transpose * v;
    return dO;
}

mat3 Angle::getJacobianTranspose()
{
    vec3 J_A = cross(vec3(0, 0, 1), endPosition - jointApos);
    vec3 J_B = cross(vec3(0, 0, 1), endPosition - jointBpos);
    vec3 J_C = cross(vec3(0, 0, 1), endPosition - jointCpos);
    mat3 Jacob( J_A.v[0], J_B.v[0], J_C.v[0],
                J_A.v[1], J_B.v[1], J_C.v[1],
                J_A.v[2], J_B.v[2], J_C.v[2]);
    return transpose(Jacob);
}
```

- **The output:**

- **Constraints for unreachable positions:**
  Defined reachable positions to be (L1-L2) to (L1+L2) where L1 and L2 are the length of upper and lower arms. Any point beyond this distance is unreachable.

- **Extra Features:**
  1. CubeMap for background scene.
  2. Toon Shading on the model for “cartoon” kind of effect.
  3. Moving Lights.

- References;

  1. https://learnopengl.com/Advanced-OpenGL/Cubemaps
  2. https://appliedgo.net/roboticarm/
  3. https://medium.com/unity3danimation/overview-of-jacobian-ik-a33939639ab2
  4. Lecture Presentation on Inverse Kinematics