# Readme File for course Project

## Part 1

**T1**. Write a parser function (call it "plyload()" ) for reading geometry info (vertices, edges, triangles) from a mesh file (in .ply) provided (see sample mesh file above). Define appropriate data structures for storing the geometry info. You should extend the menu to provide a "load mesh" button. Once it's clicked, the user can select the mesh file to load. And then your parser function should be triggered to load that file. **(done,3 pts)**

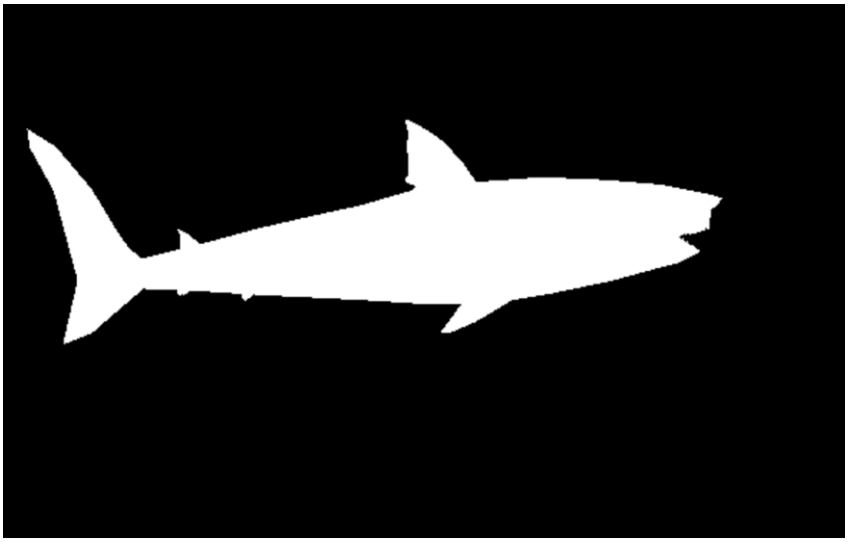
**T2**. using OpenGL commands, based on the geometry info read, visualize the mesh loaded from the mesh file. **(done ,5 pts)**

**Solution:**

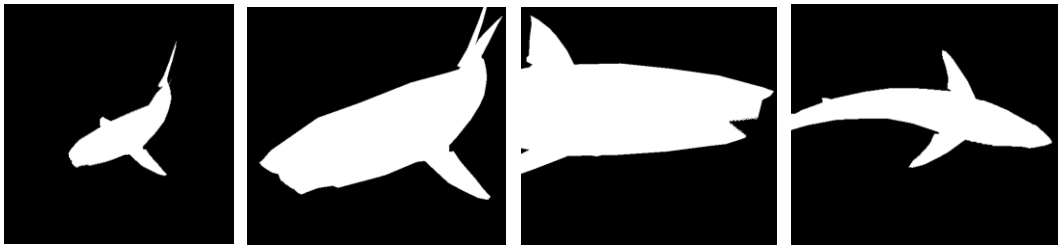
**I was not able to create a file dialog to interact with users, so I wrote a CMD program:**

```
Please Enter the PLY File Path:
C:\Users\a\Desktop\Submit\p4\sample_mesh.ply
PLY file
path:C:\Users\a\Desktop\Submit\p4\sample_mesh.ply
```

**After entering the path of the .ply file, the loader function loads the .ply file and draws in the CMD box:**



**Users are free to rotate or toggle the model pressing up, down, left or right on the keyboard. Users are also free to zoom in or out with key '-' and key '+':**

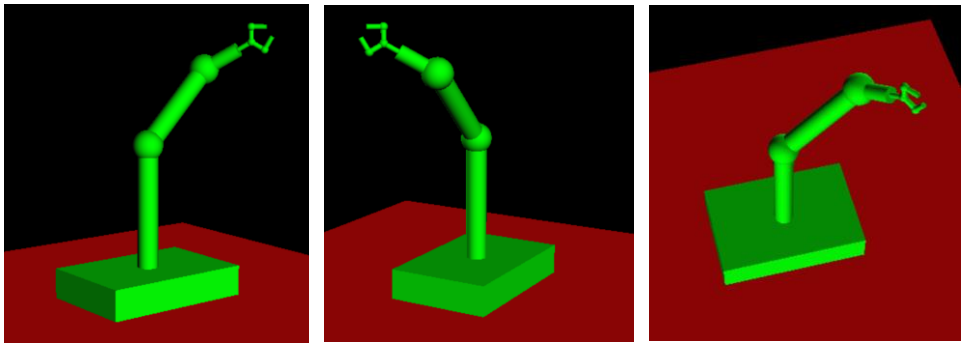


## Part 2

**T1**. You first come up with a 3D character. This character can be composed solely of primitive shapes (**box**, **generalized cylinder**, **sphere**, and **triangle**). It should use at least ten primitives and at least four levels of hierarchy. You must also use at least one each of the **glTranslate()**, **glRotate()** and **glScale()** calls to position these primitives in space (and you will probably use many of all of them!) You must also use **glPushMatrix()** and**glPopMatrix()** to nest your matrix transformations. **(done, 4 pts)**

**Solution:**

**In this task I built a Robot Arm using the framework provided by MODELER. Here is an Overview of the Robot Arm. It satisfies the requirements of the task.**



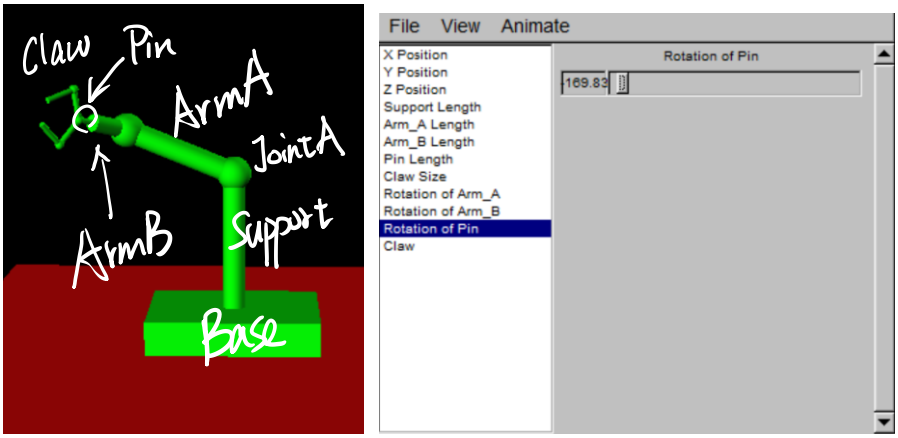
**T2.** The modeler skeleton provides functions for creating sliders and hooking them to different features of your model. You must add at least one of these as a control knob (slider, actually) for some joint/component of your model - have your character do some simple action as you scrub a slider back and forth. **(done, 2 pts)**

**Solution:**

**In this task I have added some Degrees of Freedom to the model. In the panel (pictures below) the user can adjust the parameters to change the Arm's pose.**

**Some Degrees of Freedom**

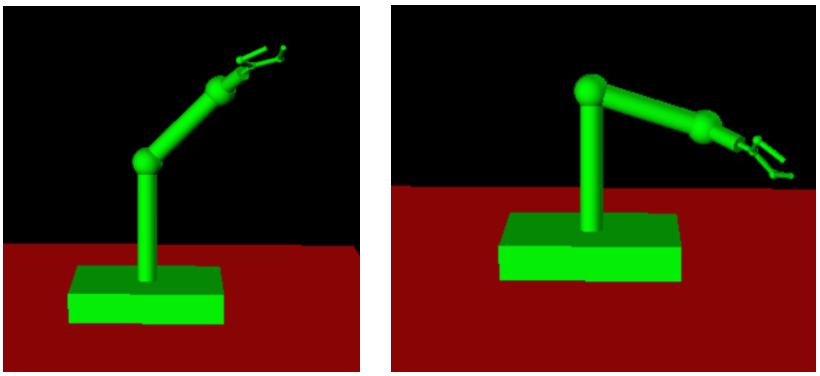
- **X, Y, Z Position**
- **Length of Connecting Rods**
- **Size of the Claw**
- **Rotation of Arm_A, Arm_B, Pin**
- **Movement of the Claw (Catch, Release)**




**T3.** In addition, at least one of your controls must be tied to more than one joint/component; this knob will change the input to some function that determines how several parts of your model can move. For example, in a model of a human, you might have a slider that straightens a leg by extending both the knee and hip joints. **(done 2 pts)**

**Solution:**

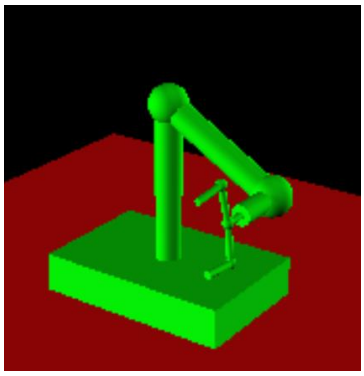
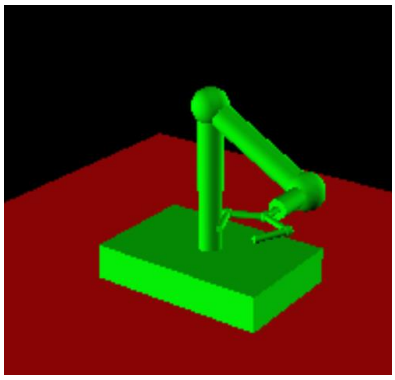
**This work has been done during Task 2. For example, a bending movement of the Arm_A would influence not only Arm_A, but also Arm_B, Pin, those Joints, and also the Claw.**



**T4.** Make an additional "animated" sequence your character can perform. Although you can try to use a timed callback "add_idle" (see here for more information), an easier solution is just to increment values each time your model's draw() function is called. If you use the menu option to turn on animation, your draw() function will be executed at around 30 times per second. You may refer to the sample modeler for a mode of the animation effect. **(done 4 pts)**

**Solution:**

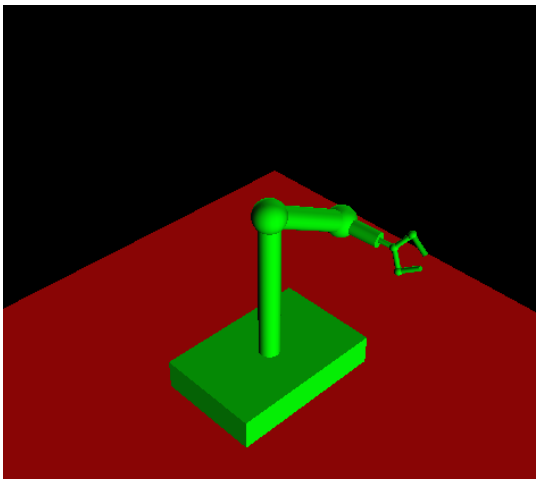
**I carefully designed an animation for the Robot Arm. Click Here to see the video**

# Project Artifact (total: 3 pts) done

**I have created images and executions to show the Robot Arm,**

**Click the Links to see. (BMP, GIF, EXE)**



## Part 3

**T1.** Find a way to represent and store the oriented bounding box and specify the variable (**2 pts**)

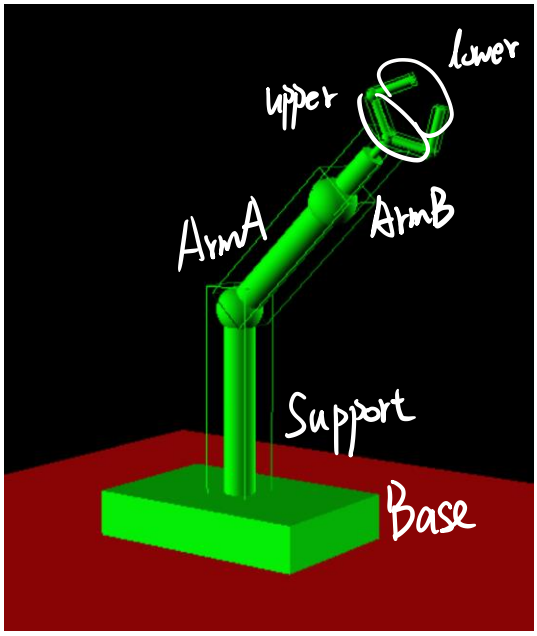
**Solution:**

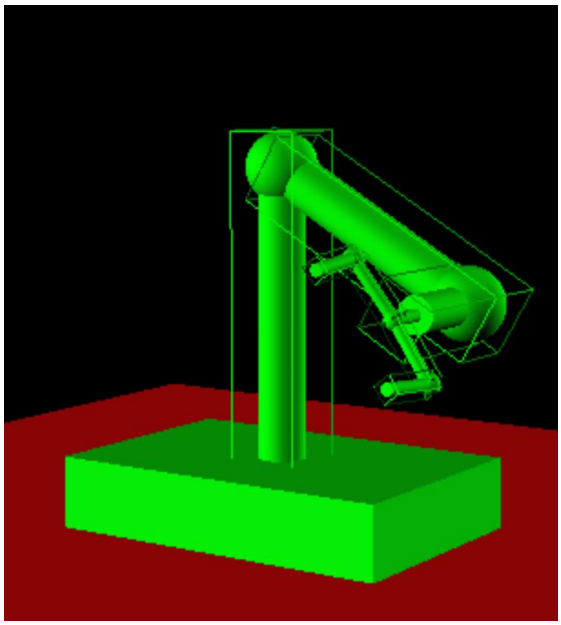
**The OBB can be represented and stored in this structure:**

- **Position tuple (px, py, pz)**
- **Size tuple (ax, ay, az)**
- **Orientation (angle) tuple (alpha, beta, gamma)**

**To illustrate how I did it with the model, I slightly modified the drawBox() function in the source file and defined a drawOBB() function to draw the Bounding Box I set for the model. (see below)**

**To make it vivid, I snapped a video clip…**





**T2.** Compute the volume of the model used in the part1 or part2 (feel free to choose one that you like to play with) ( **2 pts** )

**Solution:**

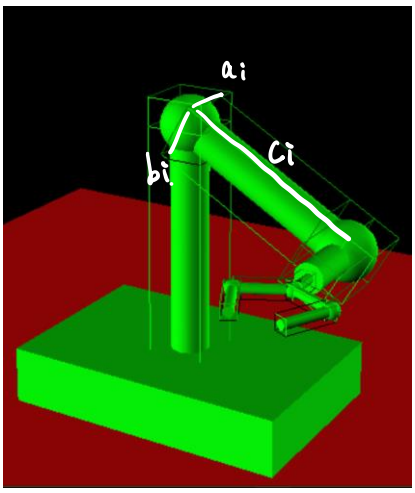
**As you can see, there are 7 OBBs for the model ( not including the Base since it is a cube itself), so the formula of the total volume should be:**

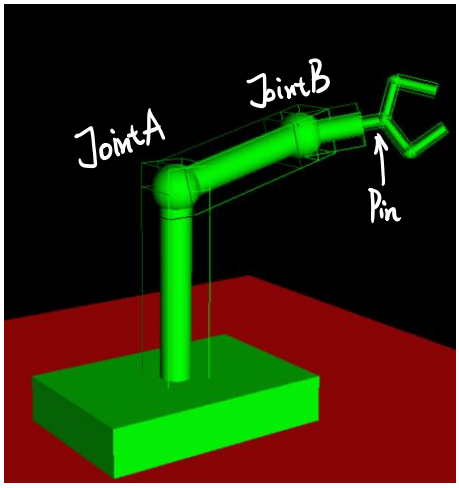
$$V_{OBB} = \sum V_{OBB_{support}} + V_{OBB_{JointA}} + V_{OBB_{ARMA}} + V_{OBB_{JointB}} + V_{OBB_{ARMB}} + V_{OBB_{Pin}} + V_{OBB_{claw}}$$

$$V_{OBB_{claw}} = \sum V_{OBB_{upper1}} + V_{OBB_{lower1}} + V_{OBB_{upper2}} + V_{OBB_{lower2}}$$

$$V_{OBB} = \sum a_i b_i c_i \qquad a_i, b_i, c_i \; are \; the \; length \; of \; sides \; of \; OBBi$$

**To make it clear, refer to the snapshots below: (OBB designed for each main components)**





**T3.** Design a cost function that describes the design goal “tight” (**2 pts**)

**Solution:**

$$\mathrm{Tight(a,b,c)} = \mathrm{V_{OBB}} - V_{model}$$

$$\sum V_{OBB_{support}} + V_{OBB_{JointA}} + V_{OBB_{ARMA}} + V_{OBB_{JointB}} + V_{OBB_{ARMB}} + V_{OBB_{Pin}} + V_{OBB_{claw}}$$

$$V_{model} = \sum V_{support} + V_{JointA} + V_{ArmA} + V_{JointB} + V_{ArmB} + V_{Pin} + V_{claw}$$

**T4.** Initialize the oriented bounding box (2 pts)

**T5.** Start from the initialization, implement an optimization solver using simulated annealing to adjust the orientation and size of the bounding box, further improving the tightness. (5 pts)

**Solution:**

**The specific formulas of the volumes of each components are given below:**

$$\mathrm{V_{Support}} = \pi * 0.2^2 * 0.3$$

$$\mathrm{V_{JointA}} = \frac{4}{3}\pi * 0.3^2 * 0.3$$

$$\mathrm{V_{ArmA}} = \pi * 0.2^2 * 1$$

$$\mathrm{V_{JointB}} = \frac{4}{3}\pi * 0.3^2 * 0.3$$

$$\mathrm{V_{ArmB}} = \pi * 0.15^2 * 0.8$$

$$\mathrm{V_{Pin}} = \pi * 0.05^2 * 0.3$$

$$\mathrm{V_{Claw}} = 4 * \pi * 0.05^2 * 0.3$$

**The specific formulas of the volumes of each OBBs are given below:**

$$\mathrm{V}_{OBB_{Support}} = {a_1}^2 * h_1$$

$$\mathrm{V}_{OBB_{JointA}} = {a_2}^3$$

$$\mathrm{V}_{OBB_{ArmA}} = {a_3}^2 * h_3$$

$$\mathrm{V}_{OBB_{JointB}} = {a_4}^3$$

$$\mathrm{V}_{OBB_{ArmB}} = {a_5}^2 * h_5$$

$$\mathrm{V}_{OBB_{Pin}} = {a_6}^2 * h_6$$

$$\mathrm{V}_{OBB_{Claw}} = 4 * {a_7}^2 * h_7$$

**Given that:**

$$\mathrm{Tight(a,b,c)} = \mathrm{V_{OBB}} - V_{model}$$

**Therefore, the definition of the tightness function should be:**

```
function output = tight(x)
    a1 = x(1);h1 = x(2);
    a2 = x(3);
    a3 = x(4);h3 = x(5);
    a4 = x(6);
    a5 = x(7);h5 = x(8);
    a6 = x(9);h6 = x(10);
```

```
    a7 = x(11);h7 = x(12);
    V_Support = pi*0.2*0.2*3;              V_OBB_Support = a1.*a1.*h1;
    V_JointA = (4/3)*pi*0.3*0.3*0.3;       V_OBB_JointA = a2.*a2.*a2;
    V_ArmA = pi*0.2*0.2*1;                 V_OBB_ArmA = a3.*a3.*h3;
    V_JointB = (4/3)*pi*0.3*0.3*0.3;       V_OBB_JointB = a4.*a4.*a4;
    V_ArmB = pi*0.15*0.15*0.8;             V_OBB_ArmB = a5.*a5.*h5;
    V_Pin = pi*0.05*0.05*0.3;              V_OBB_Pin = a6.*a6.*h6;
    V_Claw = 4*pi*0.05*0.05*0.3;           V_OBB_Claw = 4*a7.*a7.*h7;
    V_model = V_Support + V_JointA + V_ArmA + V_JointB + V_ArmB +
V_Pin + V_Claw;
    V_OBB = V_OBB_Support + V_OBB_JointA + V_OBB_ArmA + V_OBB_JointB
+ V_OBB_ArmB + V_OBB_Pin + V_OBB_Claw;
    output = V_OBB - V_model;
end
```

**It is not hard to construct a Simulated Annealing algorithm using Matlab:**

```
%%%%%%%%%%%%%%%%%%%%%%%%     Prepareation     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear all;
close all;
clc;

D = 12; % dimension of variables
Xx = [0.2*2, 3, 0.3*2, 0.2*2, 1, 0.3*2, 0.15*2, 0.8, 0.05*2, 0.3,
0.05*2, 0.3]; % lower boundary of variables
%     a1   h1   a2    a3   h3   a4    a5    h5    a6    h6   a7     h7
Xs = [0.4*2, 6, 0.6*2, 0.4*2, 2, 0.6*2, 0.30*2, 1.6, 0.10*2, 0.6,
0.10*2, 0.6]; % upper boundary of variables

L = 20; % iteration time at each temperature
K = 0.998; % K factor
S = 0.01; % step size
T = 100; % initial temperature
YZ = 1e-8;
```

**Hereby I used a random method to initiate the parameter of the OBBs, which will be optimized later on ……**

```
%%%%%%%%%%%%%%%%  Initiation of the parameter vector  %%%%%%%%%%%%%%%%
PreX = Xx + rand(1,D).*(Xs-Xx); % a random initial point
PreBestX = PreX;
PreX = Xx + rand(1,D).*(Xs-Xx); % a random initial ponit
BestX = PreX;
delta = abs(tight(BestX)-tight(PreBestX));
TraceIteration = [];TI = 0;
TraceT = [];TT = 0;
```

```
while (delta>YZ) && (T>0.01)
    T = K*T; % cooling
    for i = 1:1:L % try L extention directions at one Temperature
        for j = 1:1:D % boudary absorb
            NextX(j) = PreX(j) + S*((2*rand-1)*(Xs(j)-Xx(j)));
            if NextX(j) > Xs(j)  % out of boundary
                NextX(j) = Xs(j);
            end
            if NextX(j) < Xx(j)
                NextX(j) = Xx(j);
            end
        end

        if tight(NextX) < tight(BestX) % a best solution is generated
            PreBestX = BestX;
            BestX = NextX;
        end
        if tight(NextX) < tight(PreX) % a better(than now) solution is
generated
            PreX = NextX;
            %P = P+1;
        else
            changer = -1*(tight(NextX)-tight(PreX))/T;
            p1 = exp(changer); % accept this try though worse
            if p1 > rand
                PreX = NextX;
                %P = P+1;
            end
        end
        TraceIteration = [TraceIteration,tight(PreX)];
    end
    delta = abs(tight(BestX)-tight(PreBestX));
    TraceT = [TraceT,tight(BestX)];
end

figure;
plot(TraceIteration);
xlabel('Iteration');
ylabel('Tight');
title('Tight Varies with Iteration');

figure;
plot(TraceT);
xlabel('Temperature');
```

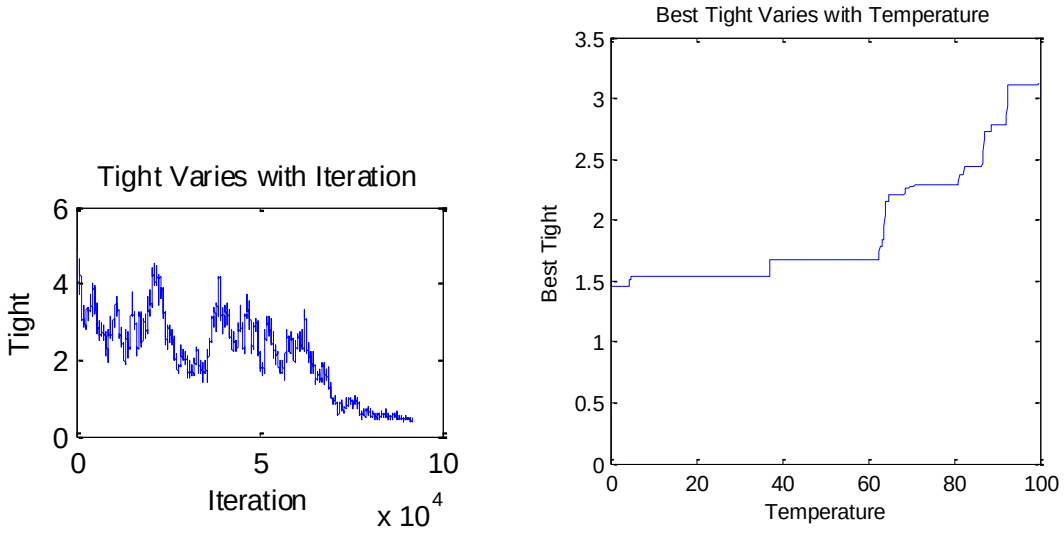
```
ylabel('Best Tight');
title('Best Tight Varies with Temperature');

[BestX,tight(BestX)]
```

**T5.** Plot how the design cost function (in the subproblem 3.) changing as the iteration of the simulated annealing increasing. (3 pts)

**Solution:**

**Using Matlab plotting functions, You can see how tightness function changes as the Temperature goes down: (SA source code, tightness function source code)**



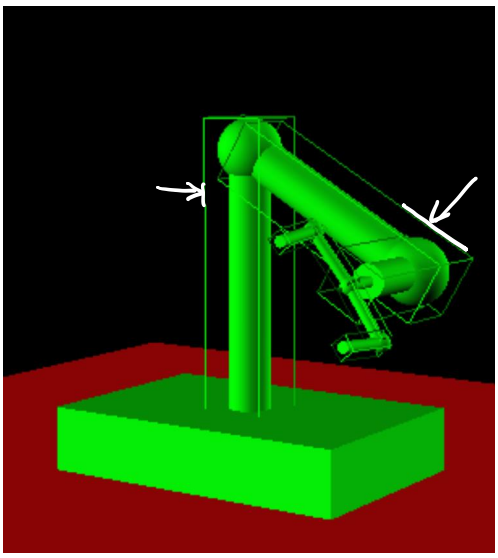

**Results of the parameter vector: (Click here for the result file if necessary)**

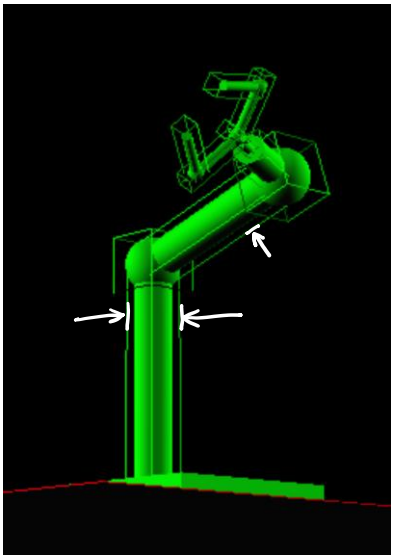
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0.4000 | 3 | 0.6004 | 0.4000 | 1.0086 | 0.6000 | 0.3029 | 0.8163 | 0.1482 | 0.5361 | 0.1674 | 0.3874 |

$$a_1 = 0.4$$
$$h_1 = 3$$
$$a_2 = 0.6004$$
$$a_3 = 0.4$$
$$h_3 = 1.0086$$
$$a_4 = 0.6$$
$$a_5 = 0.3029$$
$$h_5 = 0.8163$$
$$a_6 = 0.1482$$
$$h_6 = 0.5361$$
$$a_7 = 0.1674$$
$$h_7 = 0.3874$$

**After the optimization, the OBBs are tighter to the components:**

**Before** **After**





**To draw the frames of the OBBs, I added a function *drawOBB()* to the header source *modelerdraw*:**

```
void drawOBB(double px, double py, double pz,
     double ax, double ay, double az,
     double alpha, double beta, double gamma)
{
     ModelerDrawState *mds = ModelerDrawState::Instance();

     _setupOpenGl();

     if (mds->m_rayFile)
     {
          _dump_current_modelview();
          fprintf(mds->m_rayFile,
               "scale(%f,%f,%f,translate(0.5,0.5,0.5,box {\n", ax, ay, az);
          _dump_current_material();
          fprintf(mds->m_rayFile, "})))\n");
     }
     else
     {
          /* remember which matrix mode OpenGL was in. */
          int savemode;
          glGetIntegerv(GL_MATRIX_MODE, &savemode);

          /* switch to the model matrix and scale by x,y,z. */
          glMatrixMode(GL_MODELVIEW);
          glPushMatrix();
```

```
//glTranslated(px, py, pz);
glTranslated(-ax/2,-ay/2,0);
glScaled(ax, ay, az);
/*glRotated(alpha,1,0,0);
glRotated(beta, 0, 1, 0);
glRotated(gamma, 0, 0, 1);*/

//glBegin( GL_QUADS );
glColor3f(0, 255, 0);
glBegin(GL_LINE_LOOP);
glNormal3d(0.0, 0.0, -1.0);// face 1
glVertex3d(0.0, 0.0, 0.0); glVertex3d(0.0, 1.0, 0.0);
glVertex3d(1.0, 1.0, 0.0); glVertex3d(1.0, 0.0, 0.0);
glEnd();
glBegin(GL_LINE_LOOP);
glNormal3d(0.0, -1.0, 0.0);// face 2
glVertex3d(0.0, 0.0, 0.0); glVertex3d(1.0, 0.0, 0.0);
glVertex3d(1.0, 0.0, 1.0); glVertex3d(0.0, 0.0, 1.0);
glEnd();
glBegin(GL_LINE_LOOP);
glNormal3d(-1.0, 0.0, 0.0);// face 3
glVertex3d(0.0, 0.0, 0.0); glVertex3d(0.0, 0.0, 1.0);
glVertex3d(0.0, 1.0, 1.0); glVertex3d(0.0, 1.0, 0.0);
glEnd();
glBegin(GL_LINE_LOOP);
glNormal3d(0.0, 0.0, 1.0);// face 4
glVertex3d(0.0, 0.0, 1.0); glVertex3d(1.0, 0.0, 1.0);
glVertex3d(1.0, 1.0, 1.0); glVertex3d(0.0, 1.0, 1.0);
glEnd();
glBegin(GL_LINE_LOOP);
glNormal3d(0.0, 1.0, 0.0);// face 5
glVertex3d(0.0, 1.0, 0.0); glVertex3d(0.0, 1.0, 1.0);
glVertex3d(1.0, 1.0, 1.0); glVertex3d(1.0, 1.0, 0.0);
glEnd();
glBegin(GL_LINE_LOOP);
glNormal3d(1.0, 0.0, 0.0);// face 6
glVertex3d(1.0, 0.0, 0.0); glVertex3d(1.0, 1.0, 0.0);
glVertex3d(1.0, 1.0, 1.0); glVertex3d(1.0, 0.0, 1.0);

glEnd();

/* restore the model matrix stack, and switch back to the matrix
mode we were in. */
```

```
            glPopMatrix();
            glMatrixMode(savemode);
        }
    }
```

**To make it convenient to observe the OBBs, I created a slider for the visibility of OBBs:**