Assignment 1

CS174A Introduction to Computer Graphics - Winter 2019

Due: Friday, February 1st 2019, 1:55pm

Value: 15% of final grade

Total Points: 30

Collaboration: None permitted. If you discuss this assignment with others you should submit their names along with the assignment material. Using the code from previous offerings of the course constitutes plagiarism and is prohibited.

Submission: Submit on CCLE a zipped file (UID.zip — e.g: 398342394.zip) that includes *all* files necessary to run your submission, and (if applicable) a README file explaining which parts you only partially fulfilled or unusual instructions needed for the grader to successfully inspect your project. If you wish, you may reorganize your code into multiple files, modifying `index.html` to include them where necessary.

Setup Information: Download the files provided on CCLE and double-click `index.html` to open it in your web browser. As this assignment does not load any outside files (pictures or 3D models), running it locally will work for now, but using a local server as described in assignment 0 is still recommended.

Assignment:

Write a program using our JavaScript template (provided on the course CCLE page) that draws the butterfly scene described below. Diagrams and screenshots showing the whole butterfly or showing how the hinges should hook up to each other are included.

For drawing objects, use the template's `draw()` function on a box, a triangular prism, and a ball by replicating lines of code in the provided `Assignment_One_Scene` class which can be found in `main-scene.js`, with appropriate variations. Remember that `draw()` takes the current transformation matrix as an argument in order to place the shape where it belongs. Build your current (model) matrix out of calls to `translation()`, `rotation()`, and `scale()`.

Those functions take as arguments a `Vec` of three (3) scalars, except for rotation, which takes one (1) scalar, the angle in radians, and one (1) `Vec` object (the axis) of three floats. Only the static `Vec.of()` function can generate new `Vec`s. Here is a short example (with valid code to use with your template) employing these functions:

```
T = Mat4.translation( Vec.of( 1, .5, 1 ) );

R = Mat4.rotation( Math.PI/2, Vec.of( 0, 1, 0 ) );

S = Mat4.scale( Vec.of( 1, .5, 1 ) );

M = Mat4.identity();

M = M.times( T ).times( R ).times( S );
```

JavaScript has no operator overloading, so operations like `+`, `-`, `*`, `+=`, `*=`, etc. will not work for the provided vector and matrix types (Vec and Mat). Instead use `times()` as shown in the template code and assign its return value back into your matrix to incrementally modify it. These can be chained together:

```
M = M.times( T ).times( R ).times( S ) ;

this.shapes.box.draw( graphics_state, M, red );
```

Do this to combine your steps, and place it in between your function calls that actually draw the cubes/spheres.

Requirements:

A. You must conceptually use a hierarchical approach to model the complex objects. This means incrementally creating composite transformations by using `times( )` to multiply new terms onto the right. This allows you to benefit from intermediate values. As you progress toward the "leaf nodes" in your scene's organization (such as the antennae tips and lower legs) you can draw the rest of your shapes (all the non-leaf nodes) using the intermediate matrices. **(2 Points)**

B. Your source code must use a hierarchical approach as well (breaking up your code into a hierarchy of subroutines). Don't put all your code only into `display()`. **(3 Points)**

C. Start by drawing a horizontally stretched box in front of the camera for the Butterfly thorax. On either side, attach balls for the head and abdomen. The ball surfaces and box surface should barely touch (distance=0) and the volumes should not intersect. **(3 Points)**

D. Attach flapping wings to the thorax box. Each wing should be made of two flat diagonal boxes, and a flattened triangular prism that fills in the gap between the body and the boxes. Wings must precisely touch the thorax along its upper edge, and the contact should extend the full length of the thorax. So that the boxes do not detach from here during rotation, they should hinge at these exact points along the butterfly's lengthwise axis to flap. **(4 points)**

E. Attach legs to the butterfly and animate them opening and closing. The legs have two segments each, not one. Pieces should rotate along the butterfly's lengthwise axis, and even while they move they should stay precisely connected; the bottom outer corner edge of the upper leg segment must always contact the top outer corner edge of the bottom leg segment, like in the pictures below. Legs must contact the thorax only by a thin corner edge as shown. Leg segments rotate along the butterfly's lengthwise axis. While hinging, the boxes that make up the legs must not intersect their volumes through each other or through the thorax. These leg segments have a stretched length compared to cubes and this must not cause subsequent segments to shear. **(6 points)**

F. Attach an antenna to the head on each side, and animate them flexing under the weight of their tips. The antennae consist of nine boxes each plus a larger ball touching the ends. See the pictures below. The base (initial box) of each antenna must touch the butterfly's face somewhere by performing rotations before translating out to the surface of the head sphere. From there, be sure to translate out once more half a box length before drawing the box so that the box's bottom (not its center of mass) becomes the attachment point. **(2 points)**

G. Iteratively attach the rest of the box segments on top of each antenna base to complete each stack of nine. The top outer corner edge of each box must always be in contact with the bottom outer corner edge of the next box while they animate. Using that corner edge as a hinge point, rotate R radians outwardly for each box before moving on to the next box, where R varies over time. R should smoothly vary, yet it should also never go negative (i.e. the volumes of the box segments must never intersect through one another). **(6 points)**

H. Until now your butterfly has been sitting still, allowing you to get better camera angles to see the joints without the butterfly flying away. Now, make the whole butterfly fly in a circle around the vertical axis, centered around the world origin `[ 0,0,0 ]`. It should always be aligned with the tangent of the circle and facing forward along its path, although you may tilt the butterfly upward (make its head higher than its tail) to give it a more realistic fluttering pose. Secondly, the butterfly must move up and down while it flaps. Perform these two butterfly movements only when the variable “this.hover ” is false; this way, by clicking the “hover in place” button you can still stop the butterfly to watch its joints move if you need to build more parts onto it or adjust anything. **(4 points)**

You must rotate objects around the correct point; i.e., where they touch the parent object matters. Pay special attention to the locations of these hinges - points or edges where two boxes make contact - and center your rotations along those. You will be mainly graded on touching box corner points/edges and on the fluidity of motions.

Your scene must be qualitatively similar to the one provided, but need not match the exact motion, dimensions, or colors of the sample code. Use colors, sizes, or an added feature to customize your butterfly and make it unique so we do not get your submission mixed up with the others. In doing so, you must still adhere to the requirement details -- remember, boxes must all stay hooked up at certain corner points or corner edges.

Hints:

a. Create a method `draw_leg()` and use it for each of the legs. Call it six times, passing in a different matrix each time.

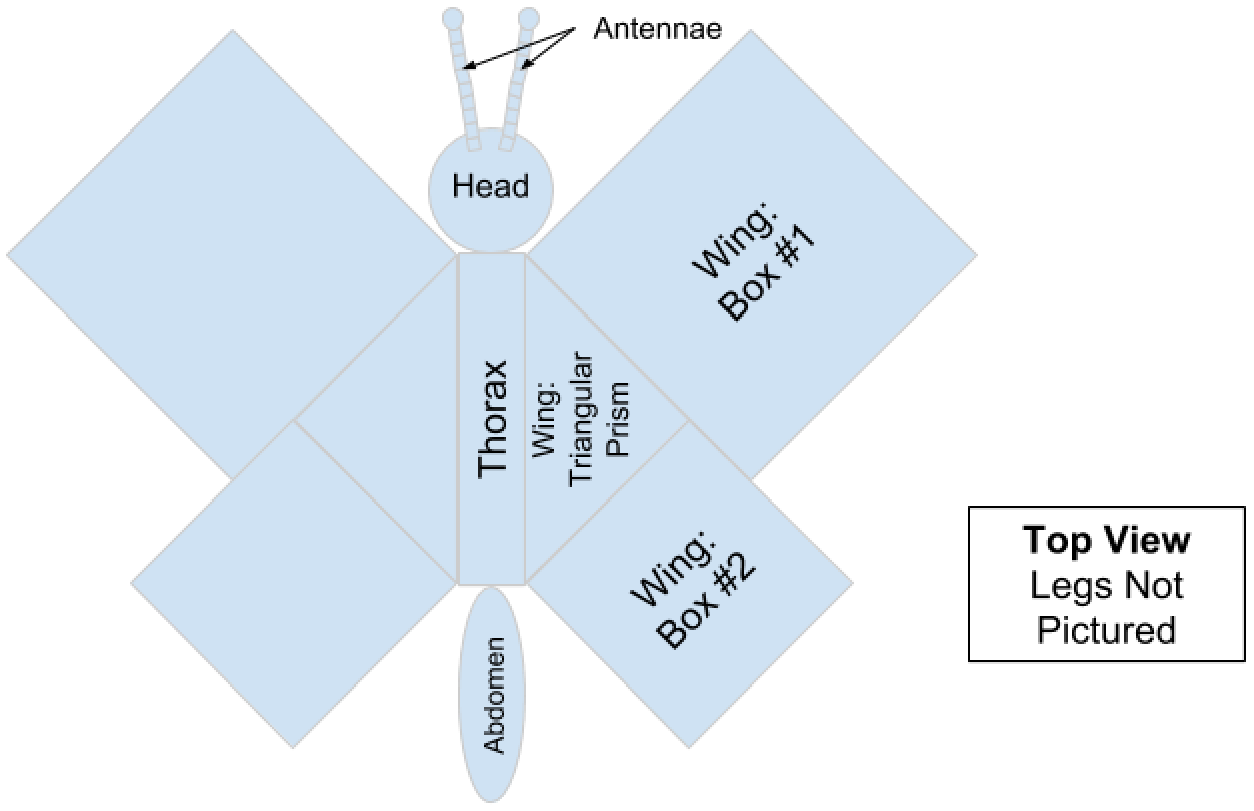
b. Your butterfly code to fill in is in index.html in a class called Assignment_One_Scene, and already contains a couple of example shapes for you to experiment with, a time variable in seconds (`this.t`), as well as controls to move the camera and control time.

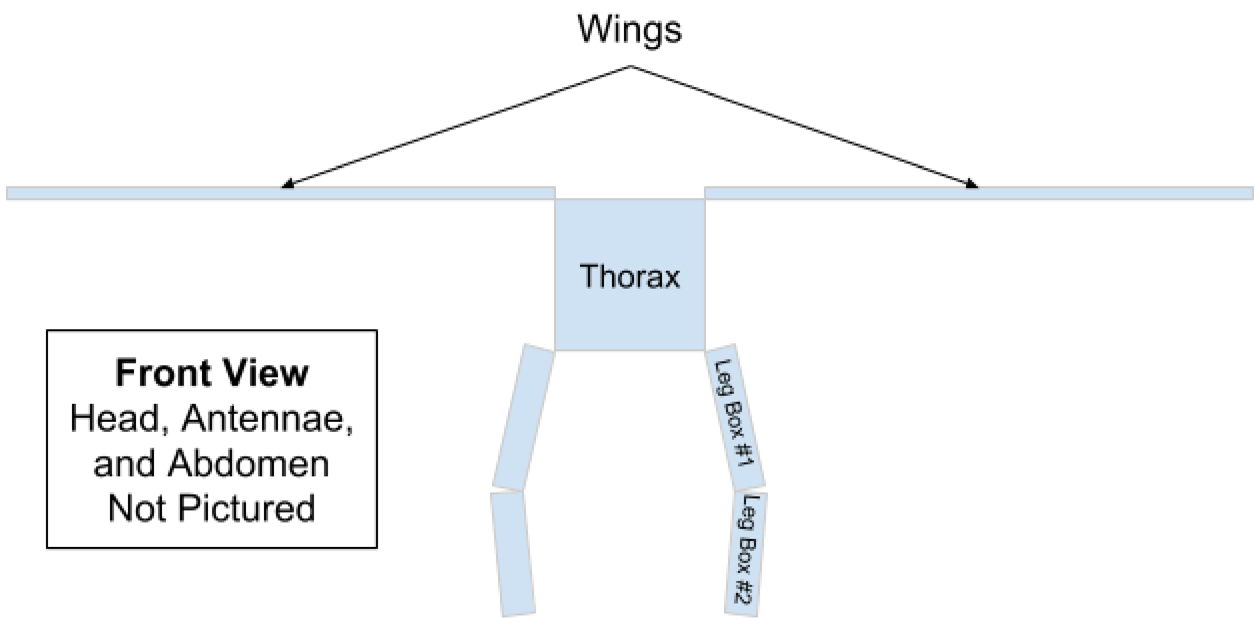
c. Functions of the form $f(t) = a + b\ sin(w\ t)$ are useful for modeling periodic motion.

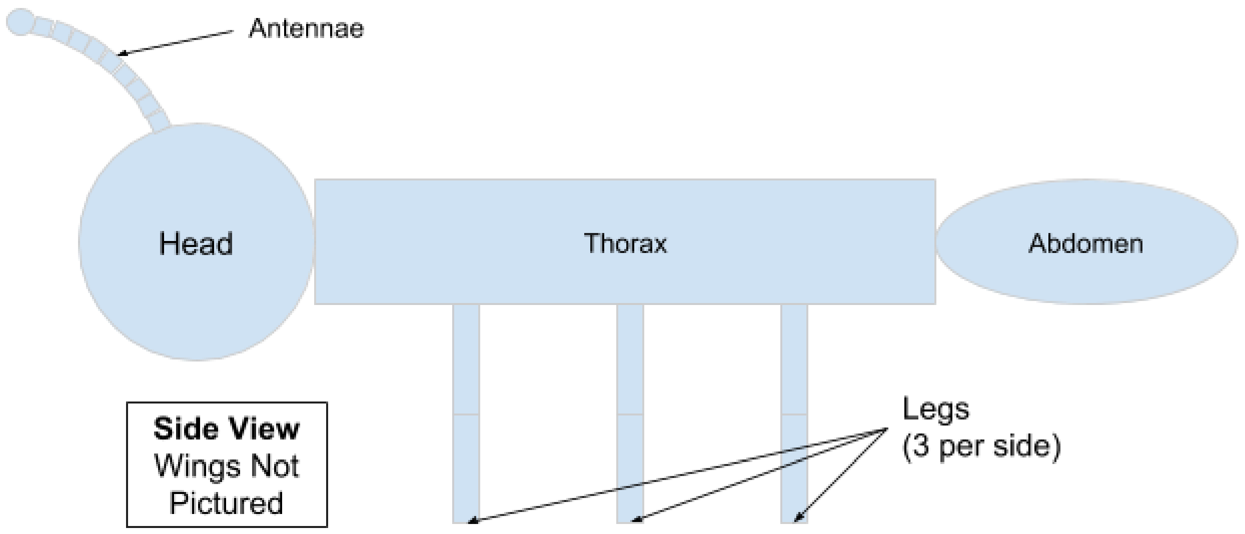
d. The resulting code for this assignment really shouldn’t be too complicated. If you’re smart about code organization, this assignment can be completed with fewer than 75 lines of code.

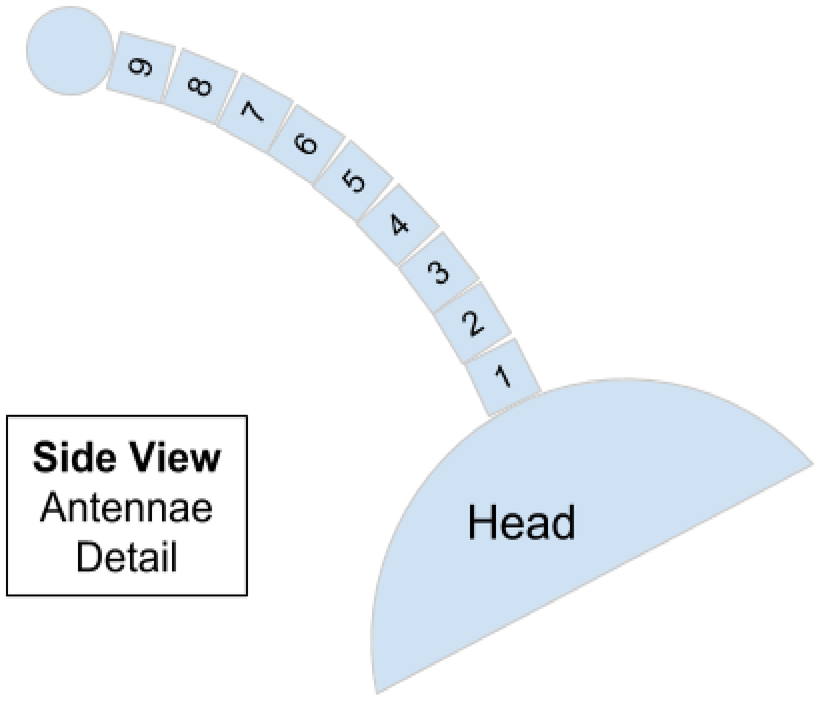
Diagrams:

The diagrams below are included to help clarify the above requirements in terms of geometry.



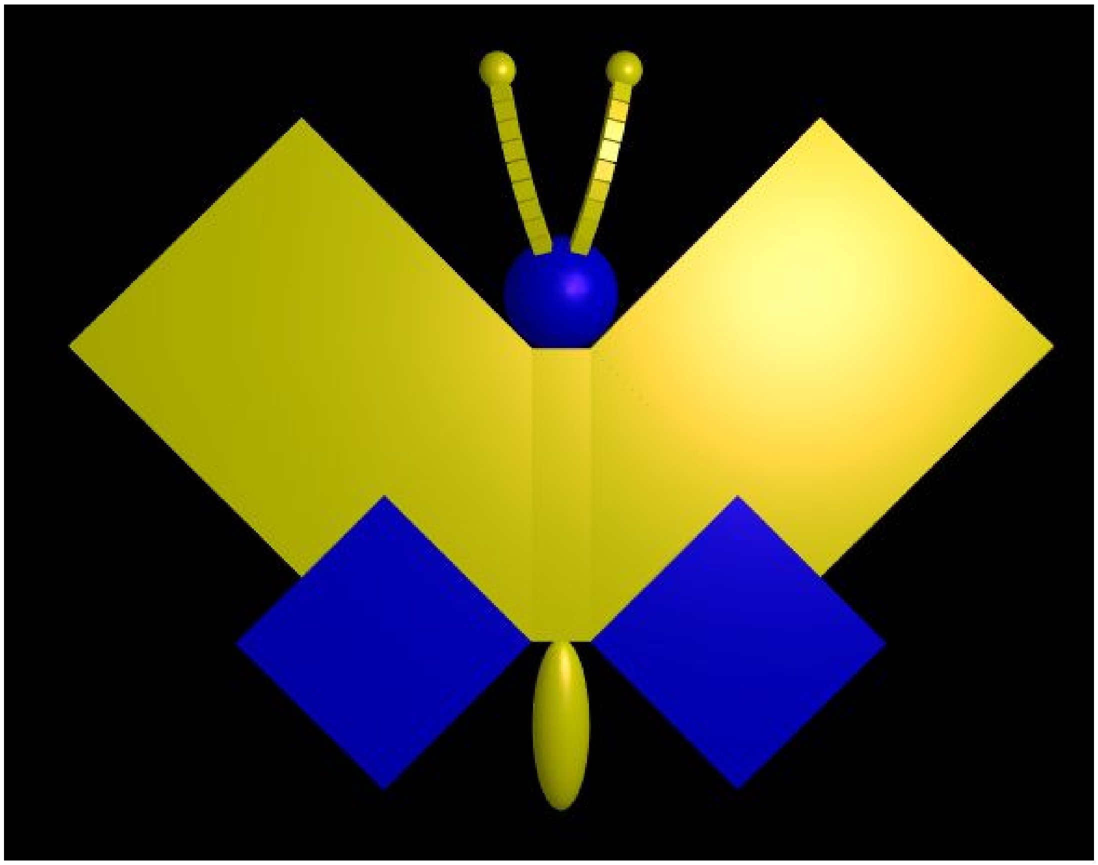



Antennae
Head
Thorax
Abdomen
Legs
(3 per side)
Side View
Wings Not
Pictured

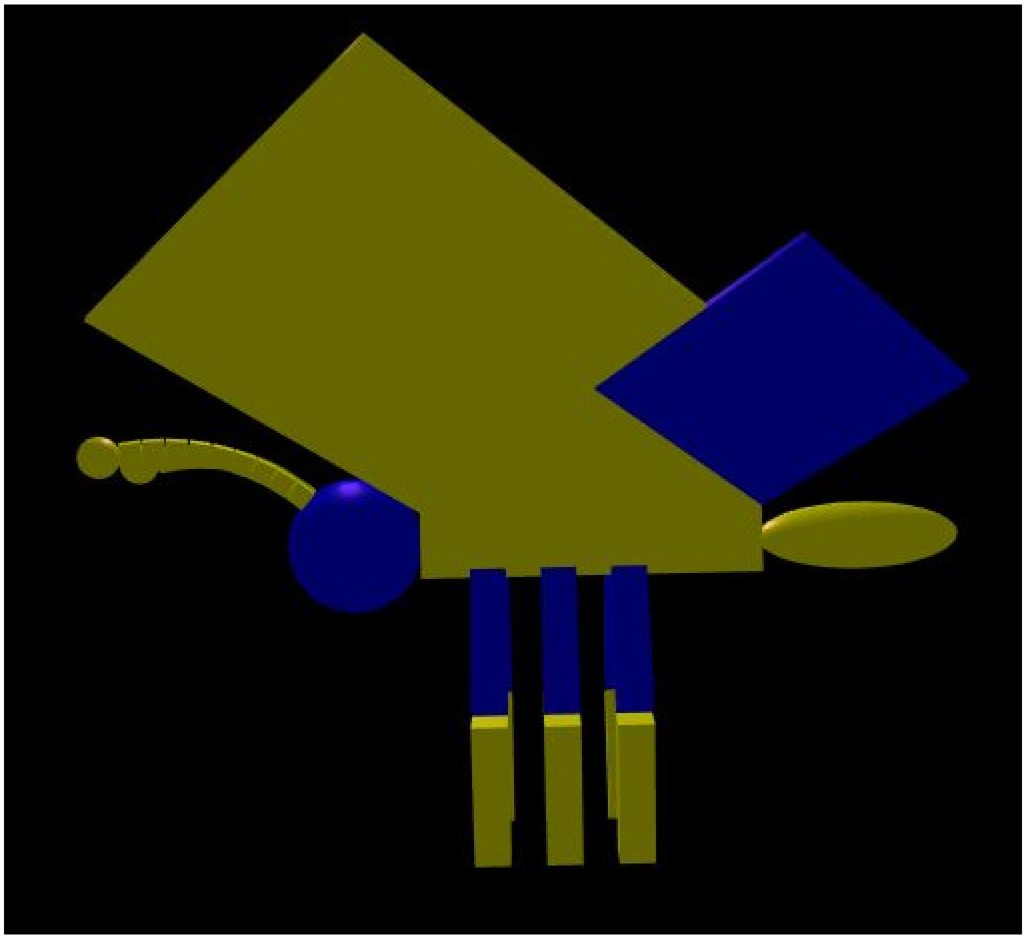

9
8
7
6
5
4
3
2
1
Head
Side View
Antennae
Detail

Screenshots From Our Implementation:

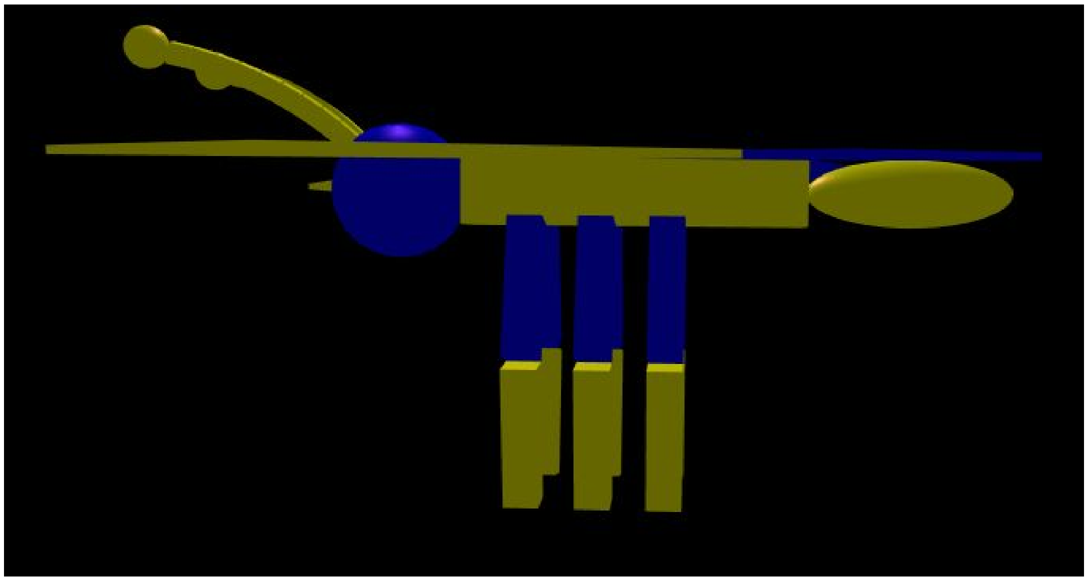
These images were taken from our implementation being rendered in the browser.



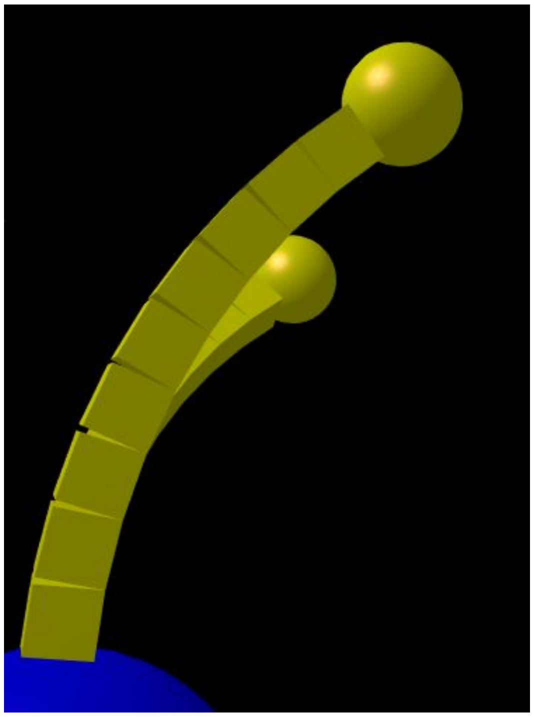
Top View



Side View (Wings Up)



Side View (Wings Horizontal)



Antennae Detail

Legs Detail