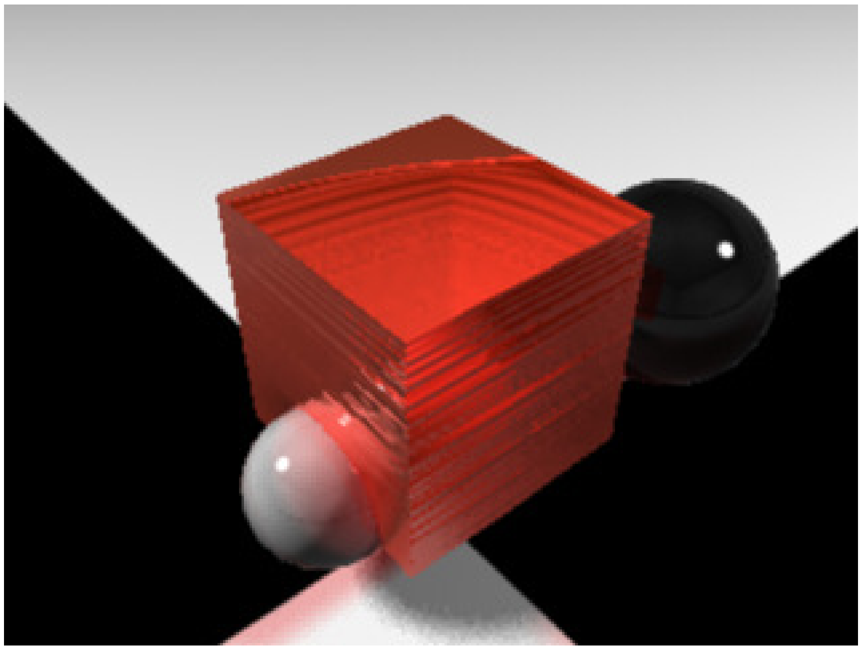
PORT JELL-O SIMULATOR INTO MAYA



A Tutorial For CIS 563

By Igor Chiang

February 10, 2011

This tutorial will provide a guide for you to port your awesome jell-o simulation from the simulator into Maya for rendering. It is composed of three basic steps: 1) setting up Maya, 2) exporting vertices in the simulator, and 3) importing the vertices in Maya. The language is in pseudo code follows

(Note: This tutorial is only for guidance; there are a gazillion ways, perhaps faster and easier, to perform the same task. This is the approach I took to tackle the problem.)

Note: This tutorial uses Mental Ray , however you may use RenderMan as well. Renderman is also installed on all the lab computers in the SIG Center for computer graphics. You have to go to Window->Settings/Preferences->*Plug-in* Manager , load the plugin if the bar is not up automatically. And you can set up a variety of shaders in Slim then render your scene using Renderman's PRman instead of Maya's built in render.

Please contact me via ichiang@seas.upenn.edu if you find any bug or anything unclear.

Setting Up Maya

It would be nice if we can just port the list of coordinate system into Maya and translate each vertex through MEL script. However, life is never easy as it can be when it comes down to Maya. The fact is that Maya uses a numbering convention that is spiral on each face. One way is to figure out the formula to derive the naming convention; sure, it will work, but I do not know and do not feel like figuring it out. So I'll teach you a little hack.

1. Create a unit cube with same number of columns, rows, and stacks as your jello.
2. Translate the cube by *<<0.5, 0.5, 0.5>>*. This will set the origin to index ijk = 0.
3. Output all the vertices into Excel or MATLAB. You should get a n x 3 matrix.
4. Make a vector of 1:n and concatenate it to the matrix.. It will serve as the mapping index function. You should have n x 4; a column of index, a column of x, a column of y, and a column of z.
5. Sort rows in the order of x→y→z accordingly. As we will loop rows, columns, and stacks in out simulator, we want to adapt the same numbering convention. Use *filter* in Excel or the *sortrows* function in MATLAB. Your index column is now your mapping function.
6. Extract the n x 1 index vector back into Maya as *int[]*.

You now have the exact mapping function to communicate with Maya. You can try to call

select MESH_Jello.vtx[$index[$ijk]];

in Maya to check that the naming convention is now the same as that in the jell-o simulator. *MESH_Jello* (*polycube*) is your jell-o mesh, is the *$index* (*int[]*) index vector, and *$ijk* (*int*) is the number of the vertex.

Exporting Vertices In Simulator

As you might have already noticed, the mesh in Maya only has the outer shell. Therefore, we only want to port the vertex locations of the outer shell into Maya. The method is very simple through the fstream library.

1. Add *#include <fstream>* to your *jelloMesh* header.
2. Construct a *void* function *outputVertices* with *int& i_frame, char* s_file* as inputs.
   a. Create an *ofstream* object *io_out*.
   b. If the frame number, *i_frame*, equals to 0, then use *ios::ate* as open mode, else use *ios::app* as open mode. This will allow you to append your consecutive frames together and start anew if you make a new recording.
   c. Create an *int* counter *i_vrt* and set it to *0*.
   d. output *i_frame* to *io_out*.
   e. Make a triple for loop in the order of rows, columns, and stacks.
   f. Inside the stack loop, the inner most loop, create an if statement such that only cases in which current row, column, OR stack equals to the boundary. (*0* OR *m_cols*/*m_rows*/*m_stacks*)
   g. Within the if statement, add the following:
      i. Get the particle position. *vec3 v_p = m_vparticles[i][j][k].position*.
      ii. Inside the same loop, output *i_vrt << " "<< v_p [0] << " " << v_p[1] << " " << v_p[2] + " ";* into *io_out*. This will be the format we will be reading from Maya later.
      iii. Increase the counter *i_vrt* by *1*.
   h. Outside of all the loops, output *endl* to *io_out*.
   i. Increase frame numer, *i_frame*, by *1*.
3. At the main function, follow the structure of already implemented screen capturing, and call *theJello.outputVertices(theFrameNum, s_outputFile)* when a specified button is pressed instead of *grabScreen*. *s_outputFile* is char* of the name of your desired output file.

You can now press the outputting button and see your result get printed out.

Importing Vertices In Maya

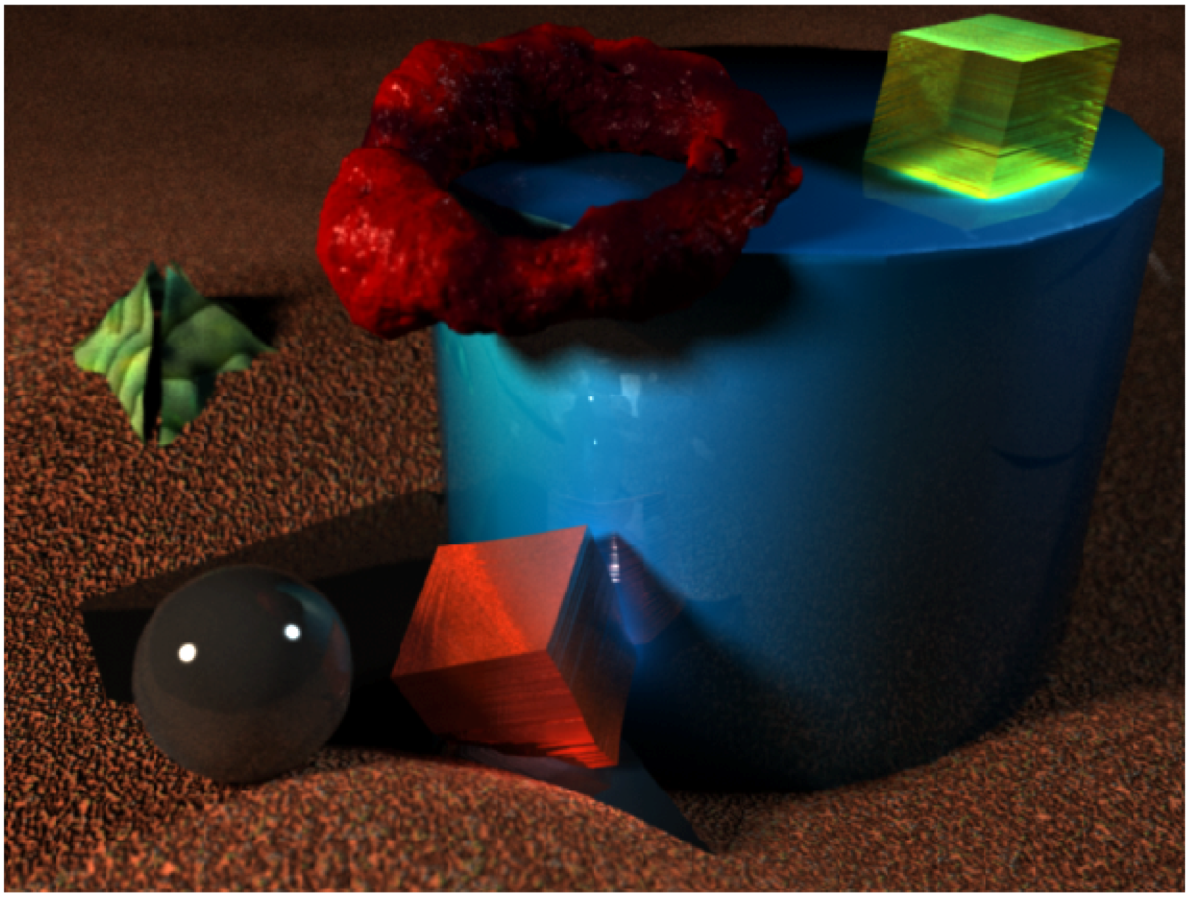
All that is left is to tell Maya to read in the file you just created!

1. Construct a MEL *global proc* with *int $i_frame_max, int $i_vrt_max, string $s_file* as inputs. These inputs are total number of frames, total number of vertices, and vertex file name you just created respectively.
2. Create a file buffer *$io_file = `fopen $s_file "r"`*. To read in a word, call *`fgetword $file`*; line, *`fgetline $file`*. As the result will be in *string* format, you will need to cast the return value into the type we need.
3. Create a *for* loop that goes from frame 1 to *$i_frame_max*.
    a. set *currentTime* to *(int)`fgetword $file`*. This will read in the frame number we outputted before.
    b. Create another *for* loop that goes from *0* to *$i_vrt_max*.
        i. select *MESH_Jello*.vtx[*$index*[*(int)`fgetword $file`*]]; The format is the same as before. This will select the vertex in order using the mapping system we created.
        ii. *move ((float)`fgetword $file`) ((float)`fgetword $file`) ((float)`fgetword $file`)*. This will translate the selected vertex.
        iii. *setKeyframe*.
4. Close the buffer with *fclose $io_file*.

OMG! You are now done importing the mesh into Maya! Add shaders and eye candies, then render using Mental Ray, V-ray, or Renderman >=]

Hope you find this tutorial useful!

This is what I came up with as of now, four days before due date:



I added:

- Custom sand shader for the ground
- Custom jell-o shader
- Random primitive objects
- Weird custom shaders (flesh and green paper)
- Depth of view
- Importon
- Causic
- Global Illumination
- Soft raytrace shadow

and rendered using Mental Ray.