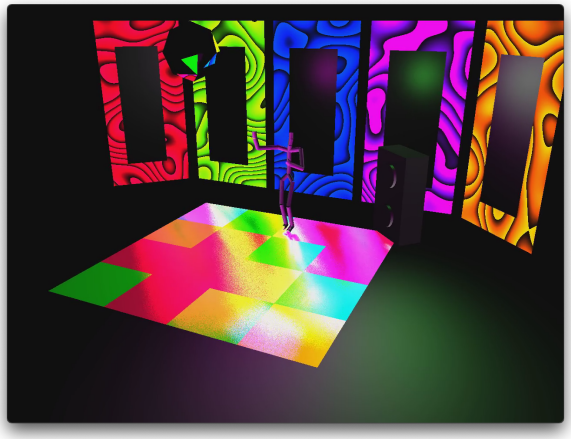
CPSC 478/578: Computer Graphics **November 8, 2023**
Fall 2023, Final Project

478 Only: You can work in pairs for the final project. However, the expectations will be higher. See §1.4 for more details.

1 The Adventures of Stick Figure

For the final project, we will build a (very) junior movie pipeline and render a 10-second movie. We perform "pre-visualization", a.k.a. pre-viz, of the movie using a simple ray tracer, and then send it off to a high-quality, distributed ray tracer for final rendering. You will post the final video to YouTube.

Here are some project results from previous years (click to see the video):



You will be designing your own sequence, but these should give you an idea of what can be done in a month.

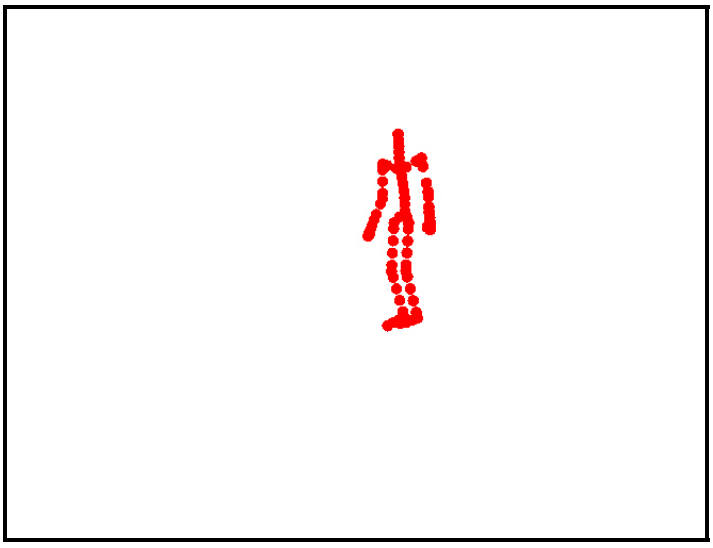
1.1 The Starter Code

Two programs have been made available to you: `previz` and `movieMaker`.

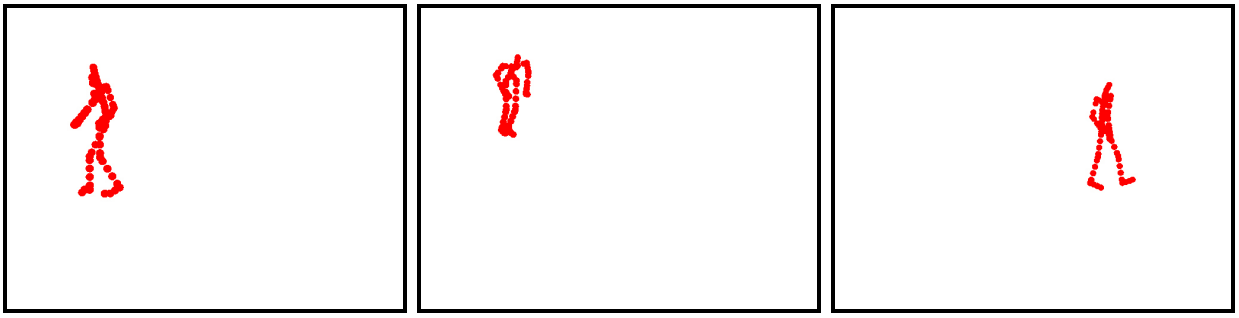
1.1.1 Using `previz`

You should be able to build `previz` by calling `make`. When you run `./previz`, it will automatically load up the skeleton file `01.asf` and the motion file `01_02.amc`, and render out a spheres-only, ping-pong ball version of Stick Figure.

The first frame looks like this,



and several of the subsequent frames look like this:



At the 300th frame of the animation, the program will automatically quit, having written out a numbered sequence frames to the `./frames/` directory. The files will be named according to the convention: `./frames/frame.*.ppm`.

1.1.2 Movie Option 1: Converting Using `movieMaker`

You have two options for building a movie out of the rendered frames. First let's try the version in the starter code, `movieMaker`. You should be able to build it by calling `make` as well. However, it has only been tested on Mac and Linux machines so **it may not build on Windows**. If this is the case, skip to the option in the next section.[1] On Mac, you may need to install `jpeglib` first using Homebrew. Once you have installed Homebrew, call `brew install jpeg`.

If the code built successfully, do the following.

[1] Another option is to `ssh` into a Zoo machine, build `movieMaker` there, `sftp` your frames to the machine, run `movieMaker`, and copy back the resulting QuickTime file.

- Copy the final program binary, `movieMaker`, to the `frames` directory of `previz`, e.g. from the `movieMaker` directory, call `cp movieMaker ../previz/frames/`.
- From inside the `frames` directory, execute `movieMaker`. It will automatically look for a sequence of frame files named `frame.*.ppm`, starting with `frames.0000.ppm`, and try to stitch them together into a QuickTime movie file, `movie.mov`.
- You should be able to play this file using IINA, the cross-platform VLC Player, or by uploading the video to YouTube. Note that it will **not** play in QuickTime 10.

1.1.3 Movie Option 2: Converting Using `ffmpeg`

If you are running on a Windows machine, or just running into mysterious build errors locally, another option is to use `ffmpeg` to convert the frames. In this case, download and install `ffmpeg`, and from the `previz/frames` directory that contains the PPM files, call:

```
ffmpeg -r 30 -f image2 -s 640x480 -start_number 1
       -i frame.%04d.ppm -vframes 1000 -vcodec libx264 -crf 25
       -pix_fmt yuv420p movie.mp4
```

The command does not fit on one line here, but you should type it in as a single line. You can again verify that the movie file is correct by opening `movie.mp4` in either VLC Player or by uploading the video to YouTube.

1.2 The Carnegie Mellon Motion Capture Database

The stick figure motion included in the starter code is from the Carnegie Mellon Motion Capture Database. There are hundreds of motions in the database, and all of them are composed of two files:

- A skeleton `*.asf` file, which contains the physical dimensions of the stick figure.
- A motion `*.amc` file, which contains the captured motion of that stick figure.

Find a novel motion in the database that you would like to use in your movie. You can load up your new motion by modifying the filenames in the `main` function of `previz.cpp`:

```
string skeletonFilename("01.asf");
string motionFilename("01_02.amc");
```

For the first checkpoint, you will need to create a movie of Stick Figure undergoing a different motion than the one distributed with the starter code.

The data is captured at a higher frame rate than we need, so you will see that the starter code skips forward 8 frames at a time. You need to generate a 10-second video at 30 frames per second, and some of the clips in the database are quite short, so find one that does not freeze before the 2400th frame.

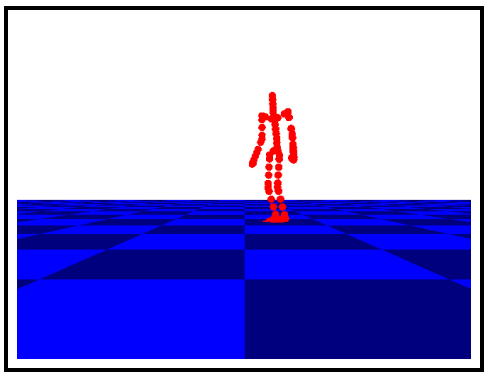
1.3 Pre-Visualize Your Scene

Let's start making the animation more substantial than just Stick Figure. The starter code provides you with a bare-bones ray tracer, which implements the sphere intersection test from the previous assignment, so you should be able to use it to generate fast previews of your scene.

Note: You *do not* have to use the ray tracing implementation in the starter code. If you like your own code from Assignment 4 better, *patch that in instead.* Working with code that is familiar and comfortable is *extremely valuable.*

1.3.1 Add Triangles

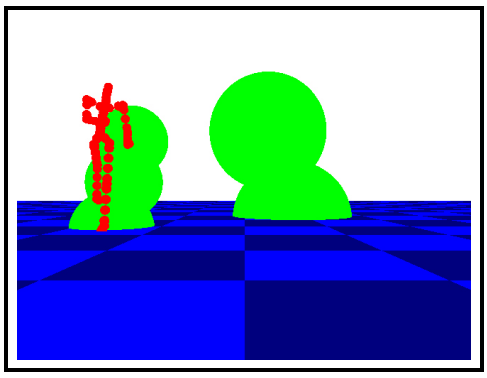
Add triangles to your scene. One obvious way to do this is to introduce a floor for Stick Figure to stand on:



Feel free to get creative here and add other exciting things that make Stick Figure's existence more interesting.

1.3.2 Add Some Obstacles

Add scene geometry to your scene. For example, Stick Figure could use some spheres to climb on instead of hanging disconcertingly in mid-air:



1.3.3 Add Camera Motion

Animations become much livelier when the camera is allowed to move, so script some camera motion. For example, have the camera follow the pelvis of the skeleton, whose translation can be retrieved via:

```
vector<VEC4>& translations = displayer.translations();
VEC4 pelvisTranslation = translations[1];
```

Again, feel free to get creative and script a motion that you think is interesting.

1.3.4 Ray-Cylinder Intersection

Ray trace Stick Figure as *cylinders*, not spheres. The endpoints of the cylinder are given in `buildScene`, at the following lines:

```
// get the endpoints of the cylinder
VEC4 leftVertex(0,0,0,1);
VEC4 rightVertex(0,0,lengths[x],1);

leftVertex = rotation * scaling * leftVertex + translation;
rightVertex = rotation * scaling * rightVertex + translation;
```

Currently, instead of drawing a cylinder, the code just lays down a line of spheres. It is your responsibility to work out the math for ray-cylinder intersection, and add it to your code.

1.4 Ray Trace The Motion

Finally, write a distributed ray-tracer for the animation you have pre-visualized, and make a movie out of the results.

1.4.1 Distributed Ray-Tracing

Add at least **two** of the following ray tracing features to your renderer, and show them off in your final movie:

- Soft Shadows
- Glossy Reflections
- Depth of Field
- Motion Blur
- Textures, either from an image or Perlin Noise
- Cook-Torrance or Oren-Nayar reflectance

At least *one* of the features must be a distributed effect (shadows, reflections, depth of field, or motion blur). **478 pairs and 578 students:** You must implement **four** of the features.

WARNING

DO THE MATH ON YOUR RENDER TIMES. If you have 300 frames, and each takes 1 minute to render, the renders will take **5 hours**. PLAN ACCORDINGLY.

If your machine has multiple cores, you can launch *multiple instances* of your program to cover different frame ranges. E.g. one core renders frames 1-150 while the other simultaneously does 151-300.

If you want to be even more aggressive, you can try rendering frames on a Zoo machine. However, you may want to get familiar with `screen` or `tmux` so that your program keeps running even after you have logged out.

If you do this, please limit yourselves to a **single Zoo machine each**, i.e. one 478 pair can use up to two machines. In the past, hoarding lots of machines to yourself has resulted in angry emails from the lab administrator. Past that, you'll have become a resource hog. Don't be that grabby person who is a resource hog.

2 Grading and Timeline

There will be two checkpoints on November 14 and 28, followed by the final submission on December 16 and a presentation on December 17 at 2 PM EST. There are a total of **40 points**, and each stage of the assignment is worth **10 points**.

2.1 Checkpoint 1: Get the Starter Code Working

The following is due at ***11:59 PM, November 14, 2023***:

- Select the motion from the Carnegie Mellon Motion Database that you would like to use in your animation. **(4 points)**
- Post a bare-bones stick-figure movie of your selected motion to YouTube. I.e. prove you got the starter code working, and successfully loaded in a new motion. *Make sure to set your video to Public or Unlisted so we can see it.* **(4 points)**
- To demonstrate each of these, turn in a `README.txt` on Gradescope containing:
  - A title for your movie **(2 points)**
  - A link to the YouTube video.

2.2 Checkpoint 2: Pre-Viz Your Scene

Your pre-viz movie is due at ***11:59 PM, November 28, 2023***. By then you should have:

- Ray-cylinder intersection **(4 points)**
- A scripted camera motion **(3 points)**

- Scene geometry, such as a floor and other obstacles. **(3 points)**
- To demonstrate these, turn in a `README.txt` file to Gradescope containing a link to your new pre-viz YouTube video. *Again, make sure to set your video to Public or Unlisted so we can see it.*

2.3 The Final Submission

Your final movie is due ***11:59 PM, December 16, 2023***. By then you should have:

- At least two of the following ray tracing effects, including one distributed effect (four total effects for 478 pairs and 578) **(8 points)**:
  - Soft Shadows
  - Glossy Reflections
  - Depth of Field
  - Motion Blur
  - Textures, either from an image or Perlin Noise
  - Cook-Torrance or Oren-Nayar reflectance
- To demonstrate these, turn in to Gradescope:
  - A `README.txt` containing a link to your new pre-viz YouTube video that *also* describes the effects you have implemented.
  - Source code that generates a representative frame from your video **(2 points)**. Include instructions on how to build and run the code in your `README.txt`.

2.4 The Final Presentation

Present your final ray-traced movie to the class during the final exam slot, ***Sunday, December 17, at 2 PM EST***.

I will have a play list running, and everybody gets 2 minutes to explain their video. Please describe:

- What effects you implemented, and where they are in the video. **(1 point)**
- What was easy to get working? **(3 points)**
- What was hard to get working, or didn't work at all? **(3 points)**
- If you did it over again, what would you do different? **(3 points)**