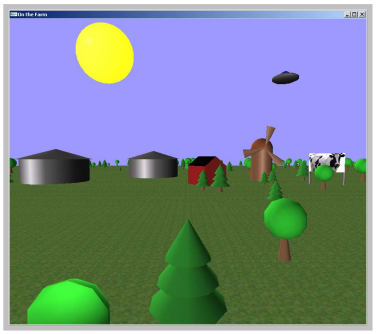
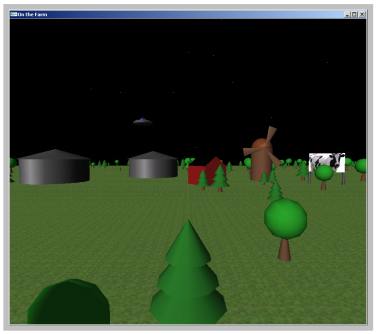
Dan Morris
CS148

Handout #12
Summer 2005



Project 3
Life on the Farm



Due: Thursday, 7/28/05, 11:59 pm

Project Goals

This project puts together everything we've worked on so far in CS148 to build a game-like virtual world. The focus is on camera control, lighting, modeling, and animation. This project also encourages you to be creative and use random stuff you find on the web, a big step toward real-world graphics programming.

Supplies

You will again use OpenGL and C++ for this project. But unlike previous projects, there is no stencil provided for you: you're building this project from scratch. We do provide tga.cpp (an image-loader) and mesh.cpp (an object-file loader), both of which were (or will be) demo'd in class, to help you. This support code is available in /usr/class/cs148/asgn/pp3 and on the syllabus page.

Demo solutions for Windows and Linux are available on the syllabus page and in the /usr/class/cs148/demos directory. The demo ships as a zipfile instead of just an executable, because it has an image file and a model file that go along with it.

Groups

This is – optionally – a group project**.** You may work in groups of up to three. Your project is expected to be somewhat more complex for group projects, although two people don't have to submit twice as much code as one person. A group will get full credit for meeting the project requirements and doing just a bit of extra credit. As a tip, a good way to parse the project into multiple people might be to have one person initially work just on supporting the different camera modes; those will likely be a significant portion of the assignment.

The Assignment: Animated Farm Scene

The goal of this assignment is to create an animated virtual world. Imagine taking a long drive down a country lane on a cool autumn day (or night), somewhere south of Sacramento.

Eventually, you will find a farm (probably several hundred). Your task is to capture this compelling image for posterity.

Your scene should include the following objects:

- A barn (a house-y structure with a roof-y top)
- A silo (a cylinder with a cone on top)
- Trees (whatever you think trees look like)
- A sky with a sun and stars (see below)
- Some object flying around in the sky (a plane, a bird, a spaceship, etc.)
- Any object loaded from a mesh file (anything but legoman).

Objects should use both diffuse and specular lighting. At least one of these objects should be textured. You can create the texture yourself or load an image. If you load an image, we recommend you use the tga.cpp file that we provide. If you want to use other code (e.g. jpeg loaders you might find on the web), that's fine, but make sure it compiles in Windows and Linux. A great image viewer that will convert just about anything to .tga is IrfanView:

http://www.irfanview.com/

Similarly, for loading your mesh file, we recommend that you use the mesh.cpp loader that we provide, but you are free to use other approaches, as long as they compile in Windows and Linux. 3dcafe.com is a great place to look for free 3d models. Many of them are not in .obj format; there are various free converters on the web; a program for viewing .3ds models and converting them to .obj format is available at:

http://techhouse.brown.edu/~dmorris/projects/winmeshview/

This is, BTW, the program that was used to create all the coke-can and lego-man figures you saw in class.

Whatever models, images, and libraries you use, please be sure to include all required files and libraries when you submit.

Additionally, your program should support the following features:

- Keystrokes should allow a transition from day to night. The sun should be visible during the day, and should move through the sky as the keyboard changes the 'time'. Stars can either "fade in" or appear instantly at sunset; they shouldn't be visible in bright daylight. As the sun moves, the direction of light in your scene should change accordingly. As the sun sets, your scene should get darker.

- You should support three different camera modes:

    - **First person:** Pressing 'a' brings you into first person mode. In this mode you can use the arrow keys to move around the scene at ground level (you don't have to be able to fly). Note that you need to use glutSpecialFunc to register a keyboard callback

for the “special” keys (like arrow keys). Up and down arrows move you forwards and backwards, and left and right turn you left and right, respectively. Think carefully about the information you need to store in order to walk around the scene (e.g., after you turn left, pressing ‘up’ should move you forward in your new direction as opposed to the direction you were facing before you turned). The “Page up” and “Page down” keys strafe left and right, which means you’re essentially sliding without turning. Don’t worry about collision detection - i.e., you’re a ghost, and you can walk through objects.

- **Scene rotate mode:** In scene rotate mode, the camera begins at a fixed vantage point looking over the scene, and dragging the mouse rotates the scene. See the sample for specifics about how the rotation should occur. You don’t have to zoom in and out in this mode.

- **Flight sim mode:** In flight sim mode, the camera is in the flying object looking straight down on the scene.

Note that dragging the mouse should only have an effect in scene rotate mode, and every time you switch out of a mode you should “remember” the current camera position for the next time you switch back into that mode.

In summary, here are the keys your project should support:

‘1’	**Time forward**	Move the day/night cycle forward a bit.
‘q’	**Time backward**	Move the day/night cycle back a bit.
‘a’	**First person mode**	See above.
up/down	**Forward/back**	Move forward and back (first-person mode only)
left/right	**Turn**	Turn left or right (first-person mode only)
pgup/pgdn	**Strafe**	Slide the camera left or right without turning (first-person mode only)
‘d’	**Mouse-rotate mode**	See above.
‘s’	**Bird’s eye view**	See above.
Escape	**Quit**	Quit your program.

Non-Farm Scenes

Clearly we don't really care whether your scene is actually a farm. If your scene is a bowling alley, a human head, or a Guns n' Roses concert, that's all well and good, as long as you include objects that satisfy the requirements. A 'barn' can be any object with planes and corners, a silo can be a cylinder connected to any other primitive, etc. There should still be a way to advance lighting through time and space, there should still be a mesh that you load from file, there should still be a texture, and there should still be something flying around that you can attach your camera to. If you're not sure if something is going to fit into your scene (e.g. stars), talk to us and we'll find something else that makes sense. We want you to be creative.

This is a great chance to make your scene something that's interesting to you. If you're really into farms, that's a fortuitous coincidence.

Extra Credit

Opportunities for extra credit abound...more complex objects or animation, creativity in scene design, fancy lighting, etc. Being able to push some objects around would be nice, allowing the "first person player" to run and jump would be nice. Allowing the first-person camera to move and rotate in three dimensions is also a challenge, since pressing *up* should still move the camera forward. We also strongly encourage you to find random code on the web to add fancy OpenGL features like shadows, fog, fire, etc. Being able to learn from online examples is a really important skill in graphics programming. Please credit any code you find; you'll still get extra credit for it, but we'd like to know where it came from. Also, in case it isn't obvious, other students' CS148 code – past or present – is not a legitimate "online resource".

The maximum extra credit for this assignment will be 20 points (out of 100).

Deliverables

All submissions are due by 11:59 p.m. on the date specified in the syllabus. Your README should specify any additional keystrokes, any extra credit you implemented, any libraries or data files that your project depends on, etc. If you worked in a group, your README should include all your group members' names and leland id's.

Submit, as always, with:

/usr/class/cs148/bin/submit