Concordia University
Department of Computer Science and Software Engineering

COMP371: Computer Graphics
TENTATIVE PROJECT DESCRIPTION
(Revision 1.3)

COMP371		**FALL 2011**
Instructor:	Serguei A. Mokhov	mokhov@cse.concordia.ca
Issued:		September 26, 2011
Revision 1.1:		October 8, 2011
Revision 1.2:		October 10, 2011
Revision 1.3:		October 18, 2011
Revision 1.4:		November 7, 2011
Part 1 due:		Sunday October 9, 2011, 23:59
Part 2 due:		Monday October 24, 2011, 13:00
Part 3 due:		Sunday November 6, 2011, 23:59
Final project due:		Sunday December 4, 2011, 23:59

1 Objectives

To commemorate 31 year anniversary of the game Pac-Man, we'll implement our own 3D version of it for this course using OpenGL and techniques learned from the course and related materials.

2 Groups

The project is to be done in groups 3-5 (all parts). Each team is provided with a group account to coordinate their group activities and manage the project's content (e.g. set up a CVS [1, 2] repository) to share documents, code and resources alike related to the group's project.

3 Procedure

Teams sequentially complete each part of the project as "programming assignments" 1–3 that constitute deliverables followed by a complete project and a report. Parts 1–3 are to be demoed to the lab instructors during the week after they are due. Whatever you demo, it must be identical to whatever is submitted to the EAS by the deadline (usually a Sunday midnight preceding the demo week). You (with the lab instructor present) or the lab instructor download the submitted code from the EAS and demo – demo need not be done the same lab days after the due date and can be done earlier if you are ready.

1. Part 1 (PA1): Modeling
2. Part 2 (PA2): Lighting and Camera Control
3. Part 3 (PA3): Texture mapping
4. Final: animation and game play, report with the software design and user manual

4 Deliverables

All parts are to be done in group. Each team member would be responsible for providing one or more models, components, etc. as subdivided by the team, and the team for the overall game environment.

1. 5% Part 1 (PA1)
2. 10% Part 2 (PA2)
3. 10% Part 3 (PA3)
4. 15% Final project and demo

In general, better quality works get better grades. The PA1–PA3 demos are evaluated by the implemented functionality in accordance with the requirements as well as the quality of the source code and modeling (procedural modeling, modular design of the code, well documented non-trivial code pieces in the form of comments, etc.).

4.1 Part 1 (PA1): Modeling.

You can start with an existing game, e.g. a possible start is [3] and gradually replace some of its components in accordance with our requirements. You can disable sound and textures at the moment. Alternatively, you may start from scratch.

In this part you are required to design a 3D scene that contains the initial configuration of the game and model the following objects using the procedural modeling techniques and OpenGL primitives. Design your code such that the objects can be easily replaceable later on.

- You have to provide the static models for:

    1. Pacman (e.g. two hemispheres and two spheres for eyes). Your team number should be "engraved" on Pacman's head top.

       For this you can model e.g. the hemispheres as triangle strips with proper normals, e.g. as in [4] or use clipping planes [5]. If you re-use that code, make sure to properly cite it and document it.

       The team number should have been modeled as cubes / polygons instead of textures at this stage. (It is to be textured in PA3.)
    2. Pellets (simple spheres).
    3. Labyrinth (4 types of walls).
    4. Floor tiles. Your team number should be present on the tiles.

       The team number should be modeled as cubes / polygons similar to a 3D Timex watch-type of digits. (It is to be textured in PA3.)
    5. Ghosts (a bunch of spheres, cones, etc.). At least 4 ghosts at the start.
    6. Scene (contains all the above).

- Use default unit size for the objects. You can scale them up and down a bit according to your aesthetic preferences.
- Place the camera 45° degrees up over the z axis backing away enough to see the whole scene.
- Allow for flat and wireframe shading. Define a key for it as you see fit.
- Allow definition of the map as an array. Later we can define levels and load different maps from a text file similarly to [3]. Let's set the default maze size at 28×28 cells.

4.2 Part 2 (PA2): Lighting and Camera Control.

The scene still remains static in this part (no animation or game play). Design and develop the following:

1. Allow for smooth shading (on top of the wireframe and flat shading from Part 1). Use the same key as in PA1 to toggle through the 3 display states. The key must be documented.
2. Model 4 street lights 4 units tall 0.20 units wide (can use cylinders and other primitives). Place them at the corners of the maze. Feel free to make reasonable aesthetic adjustments to your models.

   The *placing at the corners of the maze* part refers to the street lights' bases being **embedded** within the walls if you have a wall there, at the outside corner of respective tiles, but its base is fully within the tile's area, i.e. not "sticking out into the void" from the maze.
3. Place the four point light sources (spotlights) on top the street lights pointing down to the maze's floor such that the angles between the light direction vector $\vec{l}$ and the street light's core $\vec{s}$, as well as between the projection of $\vec{l}$ on the floor plane xz – $\vec{l'}$ and the corner floor boundaries $\vec{x}$ and $\vec{z}$ on that plane are all 45°. Specifically, such that $\angle(\vec{l}, \vec{s}) = \angle(\vec{l'}, \vec{x}) = \angle(\vec{l'}, \vec{z}) = 45°$. As an example, this means that the "light-at" point would be $(2\sqrt{2}, 0, 2\sqrt{2})$ when projected onto the floor plane assuming the light's position is $(0, 4, 0)$ and the floor is centered at the origin in the xz plane.

   Spotlight's effect should be easily visible when cast down onto the maze. Don't make it too dim. Let's standardize on the variable parameter of 25° by default for the cutoff angle.

   You can model the actual "bulbs" as yellow solid spheres.
4. Allow lights to be turned on and off all at once and individually by designated keys. Provide and document a single key to turn off all 4 spotlights at the same time in one shot.
5. Allow for ambient light to be turned on and off regardless whether the 4 other lights are on or off. Ambient light should **not** be as intense as to make spotlights indistinguishable on the maze.
6. Define material properties for the models in Part 1. Make your material definitions easily replaceable. Specifically, declare pellets to be "shiny" of variable colors, Pacman to be "diffuse" yellow, the ghosts to be "translucent" primarily light gray. Select your own material properties and properly document them for the walls and floor tiles.

Variable pellet colors means not the degree of shininess, nor that each single pellet has a color distinct from every other pellet. Let's make it a fixed range of distinct colors (e.g. white, yellow, green, etc.) that a pellet may have: between 5 to 10 distinct colors.

7. Allow to roll, pitch, and yaw of the camera with the arrow keys. Camera should be able to traverse the entire scene. You can re-use the CUGL [6] library if you want with proper attribution.

   Assuming this is the main camera from PA1, let's fix the pitch, roll, and yaw to the camera while it is stationary at some position. **This is NOT to alter the world – the models' geometry and topology should not change**, only the camera's rolls around the specified axes. The camera may move after any of the three maneuvers, usually into the look direction, but can be different moves as well (e.g. repositioning in a circle or sideway moves). If you already have a flexible good camera that does things properly you can keep it.

8. Allow 2D camera motion circling around the scene to observe the maze all around with a single key for each direction ("left" and "right"), e.g. arrow keys while in the rotation mode.

   This can be either the main camera or a new separate camera initially positioned the same way as the main camera – therefore its radius is determined from its initial position. The camera is looking at the same initial look-at point throughout its travel around the circle. This means its up, right, and look-at vectors need constant updates as the camera position circles around the scene. You can use `gluLookAt()` or alteration of your `glPerspective()` or `glFrustum()`. **Simulating this circling around with `glRotate*()` of the scene is not going to count.**

9. Place a static camera at slightly in front of each of the 4 street light sources "looking down" in the same direction as the light is cast. Allow viewing through each camera by the means of keys. Allow going back to the original camera view from Part 1.

   These are new camera objects, distinct from any cameras before this point. Keep them separate. These cameras are always stationary.

   These cameras' look-at point is identical to the street lights' light direction point, i.e. they both "look" at the same point on the floor.

   Use 'R' to reset to the original viewing parameters of PA1.

10. Place a camera in each game character, such as Pacman and all Ghosts (first person). Allow seeing through each such camera by using various keys on a keyboard as if seeing through their "eyes".

    It is prudent to place the cameras on the "forehead" of the characters rather than inside their "heads" as to not occlude your field of view of the inner head's surface :-). That is to say the first person view from the game characters should not be occluded by the characters' guts and model artifacts; it must be **either** slightly ahead **or** at exactly the same position as the model with the model NOT drawn while in the first-person view.

    These are distinct camera objects from the main and all previous cameras. They tag along with the characters at all times.

11. Make sure all the keys are documented in the help invocable by the '`H|h`' key. This is a **mandatory** requirement.

4.3 Part 3 (PA3): Texture Mapping

The primary focus of this part is configurable texture mapping of your models to be able to define different "skins" for your game objects.

IMPORTANT: the textures (their underlying images) allowed that are either of your own production, or their license allows you to use them and redistribute them (cite the source), e.g. CC images from Wikipedia [7], but check the license of the images on Wikipedia or whatever is your source, including NeHe, OpenGL.org [8], and others.

Tasks:

1. Allow for the following textures by default for your models from Parts I and II. The requirements are purposefully "loose" to allow for creativity.

   (a) A yellowish pattern-laced texture covering Pacman's outside.

   (b) A sphere-map texture covering all kinds of Pellets. Define 5-6 half-unit "mutation" or "power" pellets. Allow selective choice between texturing all pellets, large pellets only, or none at all by a key. Adjust your map loading accordingly to allow for such pellets. A single key toggling through the 3 states. Document the key.

   (c) Select a teal-colored texture pattern for the walls.

   (d) Select a light metallic-looking texture pattern for the street lights.

   (e) Select a classical/Gothic/Halloween-looking grayish translucent (with blending) texture pattern for the Ghosts (you are allowed to make them match the traditional colors).

   (f) Select rusty metallic-looking texture pattern for the floor tiles, possibly with blood stains or puddles. If you are easily frightened by the latter just the rusty pattern is fine.

2. Allow turning on and off the texture mapping (turning it off is roughly equivalent to your Part II's lighting). A single key is to be there toggling the texture mapping on and off.

3. Allow to change (alternate) Pacman's textures from the default (at least 3, including the default) by pressing a key. Optionally, one of the textures you can try for fun is a Jack-o'-lantern for the Pacman due to the Halloween season. A bonus point to those teams who do.

   Provide a single key toggling through the 3 states. Document the key.

   Bonus for the orange hollow inside of the pumpkin with the light source on the inside. The effect can be quite dramatic when the other lights are turned off, and the light glows from within the Pacman's pumpkin.

4. Don't forget your team number engravings when texturing.

5. **Start** main project report document and the `README.txt` that describe your design, procedural modeling, software architecture, and the corresponding user manual. The README part

would contain primarily the keys (the help output of your program), directory structure, and brief compilation instructions highlighting the multi-platform aspect of it if you have any.

By **starting** it is meant to have the layout, initial structure with the user manual, some screenshots, title, all the team members (only names, email addresses, NO student IDs). (This document will have to be fully completed on/before the final project submission date.)

At the demo time show the non-empty presence of the draft document of several pages with some of the mentioned material.

4.4 Final: Game Play, Levels, Shadows, Documentation

We build on top of the PA1–PA3 components. Bonus parts are optional, but will be counted towards the total for those who do them. Bonus parts, if completed, will have full weight only for those who do them from scratch instead of porting any library or piece of code from elsewhere. Most points will be checked for their degree of presence between 0 and 1.

1. Adjust your project according to all the corrections in the previous parts if you did not already have them in. Expect to demo those points.

2. Allow "zooming in" into map objects by both, (a) either controlling and moving the camera towards them or (b) by keeping the camera in place and changing the `fovy` angle. Any available camera in theory can do the `fovy` changes while being stationary, but only the main camera, and perhaps the circling one (or both if they are one and the same) can do the motion with the position change. The cameras in characters do that naturally when the characters move.

3. Full game play and scoring.

   (a) 4 Ghosts, random basic movement each time interval. They should face their direction of motion.

   (b) Collision detection between the Pacman, Pellets, Ghosts, and Walls.

   (c) At least two loadable levels.

   (d) No unreachable Pellets.

4. Mouse-enabled camera motion of the main camera (enabled by a key).

5. Allow to toggle between full screen and windowed play.

6. Eating a large pellet from PA3 causes transition of the Pacman from ordinary to super mode where its collisions with Ghosts destroy them and Pacman earns points for a period of some time. Display this transition by say converting the Pacman's appearance from normal to that of Jack-o-latern (Halloween pumpkin). You need not use that particular appearance, but any different appearance of your preference.

7. Shadows. Basic shadows are a must for Pacman, and other objects. (Ghosts naturally do not have shadows.)

8. Complete the documentation as follows:

   (a) Title the documents in the form: "A 3D Pacman Game Approach by Team X: Brief Specification, Design, and User Manual, COMP371 Fall 2011"

   (b) Do NOT include your student IDs, only your names and email addresses.

   (c) The code must be thoroughly commented, especially the non-trivial parts.

   (d) Fully complete the documentation you started in PA3.

   (e) Describe your modeling methodology, animation techniques if any, texturing, OpenGL techniques and anything else relevant. Highlight the most difficult technical aspects and challenges encountered.

   (f) Brief out your game's software architecture, design decisions. Use any tools to briefly detail that information, such as UML or any suitable kinds of diagrams and the accompanying text.

   (g) Document your modifications to any of the re-used source code components from the fellow open-source Pacman project(s), i.e. what parts are naturally yours and which came from the referenced project(s). Indicate also compoents that you did not re-use others' code and wrote from scratch.

   (h) Highlight and document all the extra things you performed above these minimum requirements.

   (i) Provide some screenshots illustrating the key features of your game.

   (j) **References to any resources used are of paramount importance.** Provide complete references section in the end.

9. **Bonus**: for those who did from scratch the **sliding maze** can be implemented to accommodate viewing scenarios when only a part of the maze is visible and if Pacman reaches half-size of the maze into the invisible part's direction, that part of the maze "slides" into the view. Those who re-used the posted code are ineligible for this bonus.

10. **Bonus**: auto game play. Let the game play by itself with all the regular game play details, simple decision making, scoring etc. While it is progressing, allow viewing from all your cameras previously defined.

11. **Bonus**: allow for adaptive level-of-detail based on the platform you are running to refine the game speed and the features to be playable at the reasonable speed. You can assign keys for that to either increase or decrease LOD. You can define LOD say as a set of booleans in an array that you cycle through with the said keys. At the end of the array all booleans are true meaning all features are enabled; at the beginning it is the most basic and fastest stuff. The indexes in the array can be set to mean to enable or disable a particular feature, e.g. you can encode gradual texturing in there of your world's objects – e.g. index X textures Pacman, Y, Ghosts, Z, for floor tiles, Z+1 for the wall textures, another index to control basic lighting vs. fully featured lighting. Define constants describing what each array index corresponds to. In your draw or display function "teach" the code to look at these indices for the decision making to enable or disable a particular LOD feature.

12. **Bonus**: allow team play with two Pacmen, but different key sets to control the two.

13. **Bonus**: Reflections. (Non-realtime and non-game play unless using the stencil buffer example). Just a frozen scene and the walls, the floor, the shiny pellets, and the street lights have the reflective properties. This bonus would count if you use environment mapping instead to simulate the reflections alongside the real-time game play.

14. **Bonus**: for those who did the game from scratch, rather than building up on an open-source project with sound, the sound effects added to the game play would be a bonus. Like with images, either produce your own sound files, or make sure the ones you get off the Internet have an open-source or similar license that allows re-distribution. Cite such sources.

15. **Bonus**: portability of the source code between Linux, Windows, and MacOS X.

16. **Bonus**: go as creative as you like beyond the minimum requirements. Feel free to enhance your models from external graphics packages you have access to, e.g. Blender [9]. The same applies to configurable visual enhancements of the textures, lights, the maze design, alteration of the default game behavior, fancier animation, and so on. All such adjustments should be clearly indicated visually, in the code, and the documentation.

5 Resources

A variety of open-source Pacman variations can be used as sources of inspiration [3, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23].Any code re-use must be properly referenced and attributed. Pay attention to the licensing. The game can further be enhanced with GLUI [24] and others. Portability is an asset.

References

[1] Dick Grune, Brian Berliner, David D. 'Zoo' Zuhn, Jeff Polk, Larry Jones, Derek Robert Price, Mark D. Baushke, Brian Murphy, Conrad T. Pino, Fred Ulisses Maranh ao, Jim Hyslop, and Jim Meyering. Concurrent Versions System (CVS). [online], 1989–2011. `http://savannah.nongnu.org/projects/cvs/`.

[2] Wikipedia. Concurrent Versions System — Wikipedia, the free encyclopedia. [Online; accessed 10-October-2011], 2011. `http://en.wikipedia.org/wiki/Concurrent_Versions_System`.

[3] r0dy. Pacman (based on Grid Crazy code, lesson 21). [online], 2006.

[4] OpenGL.org Discussion and Help Forums. Modeling a half-sphere. [online], 2004. `http://www.opengl.org/discussion_boards/ubbthreads.php?ubb=showflat&Number=209529`.

[5] Silicon Graphics Inc. GLUT examples. [online], 1997. `http://www.opengl.org/resources/code/samples/glut_examples/examples/examples.html`.

[6] Peter Grogono. Concordia University Graphics Library (CUGL). [online], December 2005. `http://users.encs.concordia.ca/~grogono/Graphics/cugl.html`.

[7] Jimmy Wales, Larry Sanger, and other authors from all over the world. Wikipedia: The free encyclopedia. [online], Wikimedia Foundation, Inc., 2001–2011. http://wikipedia.org.

[8] OpenGL Architecture Review Board. OpenGL. [online], 1998–2011. http://www.opengl.org.

[9] Blender Foundation. Blender. [online], 2008–2011. http://www.blender.org.

[10] Jeff Molofee and Contributors. Lesson 21: Lines, antialiasing, timing, ortho view and simple sounds. [online], Neon Helium (NeHe) Productions, 2000. http://nehe.gamedev.net/data/lessons/lesson.asp?lesson=21; last viewed October 2011.

[11] subtil. Pacman Arena. [online], 2003–2011. http://sourceforge.net/projects/pacmanarena.

[12] cbrianso and olerida. PANP – PANP is Not Pacman, is Net Pacman. [online], 2003–2009. http://sourceforge.net/projects/panp.

[13] Blake La Pierre, bryagh, Ravi Chodavarapu, daesuk83, jemmyc, mkopelciw, and Robert Moodey. Multiplayer Pacman game. [online], 2004–2005. http://sourceforge.net/projects/mpacman.

[14] Igor Galochkin. Ghost Commander (formerly Smart Pacman). [online], 2006–2011. http://sourceforge.net/projects/smartpacman.

[15] Igor Galochkin. Thinking 4D. [online], 2011. http://sourceforge.net/projects/thinking4d.

[16] Dan Forever and Neil Richardson. Pacman Tag. [online], 2006. http://sourceforge.net/projects/pacman-tag.

[17] Clint M. Frederickson and Mike Emery. PacMan X. [online], 2002–2009. http://sourceforge.net/projects/pacmanx.

[18] Ben Rigas. Pacman game for the Sharp Zaurus. [online], 2003–2009. http://sourceforge.net/projects/zauruspacman.

[19] sasax. Phoenix PacMan. [online], 2001–2009. Written in Java, online at: http://sourceforge.net/projects/phoenixpacman.

[20] Daniel Lélis Baggio. Winter PacMan. [online], 2006. http://sourceforge.net/projects/pacwinter.

[21] M. Babuskov and David Richmond. Njam. [online], 2003–2009. http://sourceforge.net/projects/njam.

[22] Guillaume Burlet and Clément Bourdarias. Mango quest. [online], 2001–2009. http://sourceforge.net/projects/mangoquest.

[23] Andy Lumb. PacMacro. [online], 2004. http://sourceforge.net/projects/pacmacro.

[24] Paul Rademacher, Nigel Stewart, and Bill Baxter. GLUI – A GLUT-based user interface library, version 2.35. [online], 1999–2006. http://glui.sourceforge.net/.