

CS 455 — Semester Project Proposal

Quinn Taylor & Christopher Bird (Fall 2006)

Project Summary

We plan to implement a true 3-D version of the classic arcade game Q*bert. Since the original game environment is a pyramid composed of cubes, it should be relatively easy to make the transition. We anticipate being able to implement virtually all of the original features of the arcade game and display it using OpenGL. It may be necessary to make a few simplifications in game play or complexity if they become dominating factors in the difficulty of the project, but we believe that the scope is adequate and should not need to be reduced. Because the game is old (circa 1982), we aren't counting on finding any 3-D models of the characters, so our home-brew models may end up being generic. However, we feel that this will not detract from the playability of the game.

User Interface

The user only controls the main character, Q*bert. Because he moves on a pyramid that is viewed from an angle, the controls are to move by diagonals. To begin with, we'll choose those 4 controls, but if time permits, we plan to enable the user to select other keys.

Aside from displaying the pyramid and associated characters, our interface will display the score, number of lives, level and round numbers, and the target color to which the cubes must be changed. We plan to incorporate a "death sequence" in which the user viewpoint (and possibly perspective) will change to focus on Q*bert in his death throes.

The game interface should definitely provide a way to gracefully quit at any point. Eventual improvements may include the ability to track high scores, and to save and load games.

Project Details

Game Characters

Besides the main character Q*bert, there are several dangerous objects (Coily the Snake, Ugg & Wrong-Way, and the red bouncy balls) and a few "friendly", non-threatening objects (Slick & Sam, and the green bouncy balls). It would make sense to program each of these as subclasses of a generic parent object that can occupy a cube face. This way, each could operate in their own manner (chasing, random walk, etc.) and according to their own timer.

Conceptual Movement Between Cubes

Q*bert, Coily, Slick & Sam, and the bouncy balls move from the top down, always staying on the top face of the cubes. Ugg and Wrongway move in different planes, starting on the left face of the bottom left cube (moving up and right) and the right face of the bottom right cube (moving up and left). Random movement can be achieved by bit-slicing a random 8-bit integer.

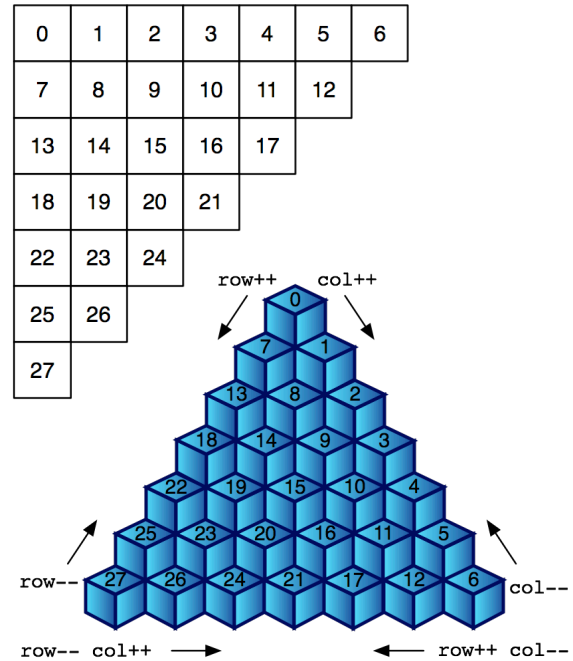


Collisions occur when Q*bert occupies the same point in space as any other object. This means he would collide with anything on top of the same cube, the right side of the cube up to the left, or the left side of the cube up to the right. (Not all collisions are fatal; green objects earn points for Q*bert.)

Storing The Conceptual Pyramid

The pyramid of cubes can be stored in a flat array of objects, indexed as shown at right. (Imagine the pyramid viewed from directly above and rotated 60° counter-clockwise.)

The idea of row and column are referenced in this context. Moving between cubes in a given plane or orientation is then only a matter of the proper combination of incrementing or decrementing the row or column, as shown in the diagram at bottom right.



The array index of a given cube is found from row and column values using this formula:

$$(row \cdot 7) - \left[\frac{row \cdot (row - 1)}{2} \right] + column$$

Each array element will store an object with information for that cube. This should include at least the following:

- Current state, which is translated to color. (Color scheme is set statically for each level.)
- Occupant(s) of each of its three visible faces.
- Pointers to possible adjacent flying disc(s), which are in the directions NW–NE–SW–SE.

Since we only care about Q*bert’s location for detecting collisions, we keep a pointer to whatever cube that is currently. That allows us to optimize collision checking, since other objects can’t collide with each other, only with Q*bert.

It seems less confusing and more efficient to store the flying discs around the periphery in pointers from boundary Cubes—rather than as “empty” periphery cubes that render as a disc—since this approach uses an array with only 28 elements (rather than 53, almost half of which would be empty) and simplifies row and column calculations and boundary checks.

Implementation Concerns

We do have a few concerns—which should be resolved in the course of the project—about implementation details that will determine how smoothly the game runs. These questions include the following:

- Should each object be responsible for moving itself on a separate timer?
- Should this be threaded, use interrupts, or something else?
- What is the desired time overlap for object collision?
- How would animation be handled?
- How will this affect frame rate and playability?