Virtual Pool (Zero)

Computer Graphics

Alexandru Victor Ștefănescu
alex.stefa@gmail.com
SN: 2128292
1/1/2011

Overview

The project consists in an attempt to replicate the billiards simulation game called Virtual Pool [1]. So far, the project implements a 3D game environment including a pool table, lamps, chairs, a cue stick and billiard balls. It allows the user to shoot the cue ball and watch the balls collide and fall into pockets. No specific billiard game rules have been implemented yet. User friendly camera controls and cue striking mechanism are identical to the original Virtual Pool.

Features:

- Real 3D models loaded from WaveFront format (.obj)
- Realistic ball physics in collisions and drag force simulation
- Realistic textures
- Additional lighting and shadows cast on balls
- Intuitive camera controls
- You can actually shoot some pool!

Design

3D Models

I have searched for a long time for a Java JOGL loader for WaveFront (.obj) files. The best one I could find was part of the JOGL Utils project [2]. However, on close inspection the code had so many issues (bugs and errors) that the current code from the project can barely be recognized as coming from JOGL Utils.

The loader supports material definition files (.mtl) for specifying material and textures for the 3D objects.

The class *WaveFrontLoader* parses the model file and the material definition file(s). The class *WaveFrontRenderer* constructs and renders OpenGL **display lists** from the model and texture data. A 3D WaveFront object can be loaded and displayed on screen in just a few commands.

Scene Graph

The project borrows the scene graph classes from Lecture 10, though they too are heavily customized. Scene graphs are used as much as possible throughout the project, though not every 3D object is a node in the graph, as container node objects are used to manage certain objects (e.g. *PoolGame* is part of the scene graph, but the contained objects, *PoolTable* and *PoolStick* are not – the rendering is managed by the *PoolGame* instance.

Scene Graph Nodes

- Perspective Camera
    - Viewpoint Marker
    - Pool Room
        - Walls, door, floor, ceiling
        - Chairs
        - Lamps
    - Pool Game
        - Pool table
            - Balls 1-15 + cue
        - Cue Stick

Lighting

The *Lamps* class manages the 3D model of the lamps hanging above the pool table. **Four OpenGL light sources** are created as spotlights falling on the pool table in the places corresponding to the light bulbs of the lamps. Additional ambient lighting added.

Shadows

Simple **projection shadows** of the balls are cast on the pool table. On the two furthest light sources are used to cast the shadows to lower the required computation load.

Physics

Collision detection between balls and with rails has been implemented. The computation model is described in [4] and [5]. Basically, during the time between each two rendered frames, the moment of the first collision which will take place is computed, balls are moved until the point of collision, the collision event is consumed, then the moment of the next collision event is computed, balls are moved, and so on. The animation should run at the same speed independent of underlying hardware.

Ball movement is slowed down by applying **drag** proportional to the distance travelled by the ball, and by simulating **kinetic energy loss** on collisions with the rails or with other balls.

No *english* effects are simulated (only straight ball trajectories – no *masse* shots, rail reflections occur at exactly the incidence angle, balls are constrained on the pool table – no jump shots, etc).

Camera & Controls

The player can pivot the camera focused on the cue ball in any direction by moving the mouse, and at any distance, by dragging the mouse (left click). The location of the cue ball can be changed by holding the move key and moving the mouse. Additionally, the pivot can also be moved anywhere on the table by holding the viewpoint change key and moving the mouse. An overview mode (viewing the pool table from the top) can be selected by holding the overview key.

The cue ball can be stricken by holding the shoot key and moving the mouse down, and then up. *The speed of the mouse coming back up is proportional to the force inflicted onto the cue ball.*

The mouse sensitivity can be temporarily amplified or decreased by pressing designated keys.

The angle of the cue stick with respect to the pool table is automatically adjusted depending on the angle of the camera to prevent seeing through the "hollow" cue stick.

Configuration

A special *Configuration* class is used to load properties files containing program settings. It offers a simple way to specify default values and constraints for each property – see class *VirtualPool*. If the settings file is missing or corrupt, it is automatically overwritten with a generated one containing the default values. This class improves greatly on the functionality of the *Properties* class of the Java SDK.

The settings available for configuring are listed below:

```
# Camera sensitivity with rspect to mouse movement
game.camera.sensitivity = 0.2
# Camera sensitivity temporary adjustment factor
game.camera.sensitivity.adjust = 3.0
# Perspective camera Field Of View [realistic=45]
game.fov = 45.0
# Control how fast balls are moving
game.speed = 1.0
# Sensitivity and hit force amplify trigger
keys.adjust+ = Period
# Sensitivity and hit force soften trigger
keys.adjust- = Comma
# Enter aiming mode
keys.aim = A
# Move cue ball
keys.move = M
# Show table overview
keys.overview = X
# Pause game
keys.pause = Space
# Quit game
keys.quit = Escape
# Respawn balls on table
keys.reset = R
# Enter cue stick strike mode
keys.shoot = S
# Change viewpoint
keys.viewpoint = V
# Kinetic energy loss coefficient
physics.elasticity = 50.0
# Drag coefficient
physics.friction = 300.0
# Force of cue stick strike
physics.hitforce = 1.0
# Strike force temporary adjustment factor
physics.hitforce.adjust = 3.0
# Enable line antialiasing [true/false]
system.antialiasing.line = true
# Enable point antialiasing [true/false]
system.antialiasing.point = true
# Enable polygon antialiasing [true/false]
system.antialiasing.polygon = true
# Target number of Frames Per Second [1-100]
system.fps = 100
```

Misc

- Logs for monitoring 3D model loading are created at each run
- Actual frames per second counter in lower right corner

Features I will one day implement

- Reflective surfaces on balls – environment mapping
- Realistic ball texture position with respect to ball movement
- Bump mapping on floor and table cloth
- Volumetric lighting coming from the four spotlights
- Complete ball physics (english, masse, jumping)
- Actual billiards rules and games + AI for playing against the computer
- …

User Manual

- MOVE MOUSE – look around
- LEFT CLICK & DRAG – zoom in/out
- HOLD V & MOVE MOUSE – change viewpoint
- HOLD M & MOVE MOUSE – move cue ball
- HOLD S & MOVE MOUSE UP-DOWN – strike cue ball
- HOLD X & MOVE MOUSE – table top overview
- PRESS A – return to aiming mode
- PRESS < (Comma) – activate/deactivate sensitivity softening
- PRESS > (Period) – activate/deactivate sensitivity amplification
- PRESS Space – pause game
- PRESS R – reset balls on table
- PRESS Escape – quit program

Loading time: 1-2 seconds on my machine. Program should start right up. If not, please give it another chance ☺

References

[1] Virtual Pool 3 DL, Celeris 2007, http://www.celeris.com/games/vp3/

[2] JOGL Utils, https://github.com/sgothel/jogl-utils

[3] JOGL Demos, https://github.com/sgothel/jogl-demos

[4] Compsci, http://compsci.ca/v3/viewtopic.php?t=14897

[5] NeHe Lesson 30, http://nehe.gamedev.net/data/lessons/lesson.asp?lesson=30

Screen Shots