CSC 165 – Midterm Study Guide

Game Design Vs Game Architecture: Game design is how a game looks and plays; game architecture is how a game is built. In game architecture, the goal is to make use of and work with a game engine. In game design, you are concerned with making a beautiful game that is fun to play.

"Genres", "themes", and "dimensionality": Game genres include action, adventure, RPG, FPS, etc. Game themes can include wizards, medieval, aliens, sci-fi, sports, etc. Game dimensionality is concerned with player motion (0D, 1D, 2D, 3D), object/NPC motion, view/camera motion, and how the ground/space is perceived.

The role of a game engine: A game engine is a reusable collection of modules that are independent of any game logic, encapsulate platform dependences. It exists to allow for the creation of a "good game" without having to be concerned with many low-level details.

Tightly coupled game loop: A series of step that the computer goes through to create every frame. In a tightly coupled game loop there is three steps: Handle player input, update game logic, and render the scene.

Input devices and Controllers: Input devices range from buttons, POV hat switches, axis's, D-pads, triggers, etc. These are arranged on controllers such as gamepads, keyboards, mice, and joysticks for ease of use. A controller is sometimes called a device collection.

Device abstractions "button", "key", "axis": There are two fundamental device types a button (gamepad) or key (keyboard) which have two states, pressed, or not pressed. These are frequently represented as 1 or 0. The other type, an axis, returns a float. There are two types of axes. A continuous axis which returns a value in a range and a discrete axis, which returns a value in a set. These can be absolute or relative.

The role of an input manager: To simplify input managing. Allows for the association of "actions" to input devices/controllers. It manages the input-event mechanism, taking care of the event queues (where actions are queued and pulled from when being handled). Each controller's event queue is shared into an event queue in the manager, where all the input for every device can be handled.

Setting up an action for handling input events:

1. Register a user-specified action corresponding to the given controller and component

   associateAction(controller, component, Action);

2. Poll the underlying device event queue's and perform event dispatch every update cycle

   im.update(float time);

Modifier Actions: Where an Action state modifies another action's behavior. To setup we pass the modifier action to the behavior action. When the modifier action is invoked its state changes. When the behavior action is invoked, it checks this state (from the reference object) and changes its behavior accordingly.

Homogenous 3D coordinate for points, vectors, and matrices: The process of adding another dimension to a coordinate. This helps in mapping a 3D world to a 2D projection. Allows for the execution of several useful matrix operations and transformation (translations, rotations, scale, projective, etc.). Homogenous coordinates can be multiplied by a scalar and can define an infinitely far away point (gives direction).

Specifying a camera with position and UVN axes: The position of a camera is defined as a point in 3D space. This defines the location of the camera. The orientation of the camera is defined by the UVN axes which denote yaw, pitch, and roll, respectively when we rotate around them. Note, the UVN system is a left-handed coordinate system. These axes must stay orthogonal to each other or weird things happen.

Specifying a view frustrum: The view frustrum of a camera is specified via the far-clipping frame (no objects beyond it are rendered in view) the near clipping frame (no object within it are rendered), the projection plane (where the world is rendered on a 2D plane). World object within this defined space are rendered in the camera's perspective. In addition, FOV and aspect ratio are defined to specify the view volume. The space encompassed bounds are the view frustrum and is the region of space that may appear on screen.

Basic Camera Manipulation (pitch, yaw, roll), look-at:

- New location: NewLoc = CurrentLoc + (viewDirVector (N) * moveAmount)
- Yaw: rotate U (side-axis) and N (view direction) around V (vertical axis): Rotate(U, V); Rotate(N, V);
- Pitch: rotate V and N around U: Rotate(V, U); Rotate(N, U);
- Roll: rotate V and U around N: Rotate(V, N); Rotate(U, N);
- Look-at:
    - N = targetPos – cameraPos
    - U = N X worldUp(y)
    - V = U X N

Local Vs Global Yaw: In local yaw the object is rotate around its local up axis, while in global yaw the object is located around the worlds up (y) axis.

Specifying an object's vertices and vertex attributes (position, texture coordinates, normal vectors): Three vertexes make up a triangle in 3D space. The position is defined from the origin and is mapped textures based on texture coordinates associated with a 2D image (0, 0: Lower left, 1, 1: Upper right). Each vertex is assigned a texture coordinate, which is interpolated for all pixels when the object is rasterized. Each vertex is also assigned a normal vector (perpendicular to the surface defined) that is used for lighting.

Rasterizing of vertices and their attributes: The process of filling in all the space of each defined triangle with pixels. Based on raster lines or rows of pixels on the screen. The points along each raster line are determined (between each defined vertex) then points along the raster lines are determined. The color of these pixels is interpolated from the color of the defined vertexes or texture coordinates of the defined vertex

How texture images are applied based on texture coordinates: A texture use a basic coordinate system with (0, 0) defining the lower left of the texture and (1, 1) defining the upper right of a texture. A vertex is assigned a position on this texture allowing a "slice" to be cut to be applied to the texture. Rasterization interpolates the texture coordinates for the plane as it is rasterized.

Specifying indexed vs non-indexed vertices: A non-indexed vertex is where each vertex is individually defined, regardless of whether or not they share a point with another vertex. This allows for defining more unique colors/textures but takes up more space. An indexed structure is where vertexes that share a spot are only defined once. Allowing for less storage space, but little control over color/texture.

Graphics-pipeline and its relationship to rasterization: The modern graphics system uses shader programming in a pipelined architecture. The application code (game) goes through the OpenGL API, which sends the vertices to the graphics card and is processed through a pipeline consisting of vertex processing, primitive assembly, primitive processing, rasterization, pixel processing, fragment testing, and frame buffer. This code runs on the GPU (through shader code in GSL) and not the CPU

Homogenous translate and scale matrices: Translation transform uses a 4x4 matrix that has 1's down the diagonal with the X, Y, Z translation amounts on the right. This moves the object by a relative amount. The Scale matrix uses a 4x4 matrix with the scale factor along the diagonal.

Rotation with Euler Angles: Rotation in 3D requires translation to/from the origin. This can be broken down into 3 rotations around the X, Y, and Z axes. The Euler angles are these rotations around each action that accomplishes the desired rotation. After performed, the rotated object is translated back to its original position.

The purpose of a perspective matrix: A matrix transform that is not affine. It is used to render a scene into perspective. Requires the frustrum def (FOV, aspect ratio, near & far clipping plane) and computes a perspective matrix based on what/how it is scene. This is generally used by the game engine renderer.

Factor used to specify ambient, diffuse, and specular components of ADS: Ambient is the amount of low-level illumination that equally affects everything in the scene. Diffuse is the brightness of the object based on the light's source angle of incidence. Specular conveys the shininess of an object by strategically placing highlights where light is directed mostly toward the eyes (camera).

Specifying light types: Point light is a light that has a location and intensity and emits light in all directions. Directional light is a light with no position but a direction in the way that it hits the scene. A spotlight has a location, direction and intensity and is very bright where its pointing. Each is defined by RGBA components.

- Ambient is calculated my multiplying the light and material ambient to get the observed ambient (multiply red, green, and blue) together.
- Diffuse considers the angle of incidence. Calculated using the light and material diffuse components multiplied by the cos(angle of incidence). Dot product of the normal and Light direction vectors.
- Specular computation depends on the ingle of incidence and the vector pointing toward the camera. This angle tells us how bright this specular component should be. Calculated by multiplying light and material specular times the angle of incidence raised to the shininess value specified in the material.

Shininess modelling in ADS: The shininess of a material is specified to allow for the calculation of the size of specular highlights on the surface of an object.

Material specification role: The ADS components of a material tell the rendered how the object reflects and is affected by light. Without these components it would be impossible to calculated how the light affects the scene. Calculating how light interacts on an objects surface is dependent on both the light and the material.

6DOF Vs Constrained Cameras: In 6DOF cameras allow the user to move the camera in any combination of roll, pitch and yaw. However, this introduces some strange effects where the orientation changes. In a constrained camera, you can use global yaw instead of local yaw to eliminate this effect. A constrained camera simply puts some limits on the camera's movement.

Specifying a Mouse-Look controller: Inputs from the mouse can be captured one of two ways, through an input manager's axis devices and through a window manager's mouse listener routines. Challenges with the latter include recentering the mouse after every moment to prevent it from disappearing from off the screen.

Chase cameras and the concept of a spring system: A chase camera follows the avatar by maintaining a constant relative view (over the shoulder, behind the back) of the avatar. It is typically placed on "springs" to reduce the jerkiness of a camera as the avatar moves.

Orbit Camera Controller: This controller allows the user to adjust the azimuth (rotation around the target) elevation, and radius (distance from target). This camera controller stores these values along with the target node and computes the camera's location based off the target's position using spherical coordinates.

Factor's for setting up viewports: A viewport is setup by first defining its dimensions. Based on these dimensions RAGE changes the view frustrum's aspect ratio to match the viewports aspect ratio. These viewports are then displayed on screen.

What is Full Screen Exclusive Mode and how is it setup: FSEM allows the program to take up the entirety of a screen and maintain exclusive control over it. To setup in java we need to <u>setResizable() to false</u>, <u>setUndecorated() to true</u>, and <u>setIgnoreRepaint() to true</u>. Also we need to pass -Dsun.java2d.d3d=false to the JVM.

Concept and uses of the scenegraph: Allows for the game world to be defined hierarchically. The scenegraph is a tree where the root node is the scene. The renderer traverses this tree as it renders the scene. hierarchical nodes inherit certain properties from their parents (concatenating parent's world transforms with child's local transforms to get child's world transforms).

Scenegraph traversal & maintaining world transforms: Every node in the scenegraph has both local and world transforms. The local transforms are concatenated with parent's world transforms. These are saved as this individual node's world transforms. When the renderer updates the picture, only renderable (generally leaf nodes) are visited. When an update occurs to an object's world transforms, it is trickled down to its children. A flag is used to determine if an update is required.

Node controllers & incorporation into the scenegraph traversal: Node controllers affect the local transforms of an object. Therefore, the node controllers' changes are calculating before updating the world transforms with the node's parents. After this is done, all the children under the node are recalculated in the same manner.

Uses for scripting in games: Scripting can be used for Gameworld creation & initialization. Dynamically modifying game details, providing user defined functions (mods) modifying player/non-player characters. Game settings, features, and testing.

Executing a script and script function: Executing a script requires embedding an interpreter in your game code (we will use Nashorn). This is passed, and loads an external javascript file. The script engine maintains a table of values associated with names. These are filled in by the running script. In JS the <u>last value calculated is returned</u> to the invoker. A function is invoked by casting the scripting engine to an invocable object. Then calling invokeFunction("function name", argsList)" in a try catch.

- engine.eval() is called to run the script
- engine.get("variable name") is called to retrieve a variable from the created table
- invocableEngine.invokeFunction("fName", arg) calls a function, arg is an array of parameters.

Fat client Vs Thin client: A <u>fat client</u> is where each client runs their own world simulation (Issues: duplicated code, sync across client worlds, clients must be deterministic. Advantages: Fast local updates, server handles only msg switching). A thin client is where the server runs the world simulation. (Issues: more network traffic, server bottleneck. Advantages: clients can be non-deterministic, sync is much easier).

Concept of ghost avatars: Ghost avatars are player or non-player characters that are controlled by a "ghost controller" that receives updates from a "communication controller" that directly gets updates from the server. It is a way to portray other players on the network in the local client's Gameworld.

Simple network protocol for games:

- **Client:**
    - Client can send create(name, pos) to inform the server of a new participant.
    - Client can move/rotate(senderName, amount) to inform the server about a change.
    - Client can send detailsFor(addr, port, pos, orient) as a response for a wantsDetails()
    - Client can say bye(name) which informs the server the client is leaving.
- **Server:**
    - Server can create(name, pos) inform clients of a new remote avatar.
    - Server can move/rotate(senderName, amount) inform client of a change in a remote avatar.
    - Server can send wantsDetails(addr, port) which informs a client that another client wants an update.
    - Server can send detailsFor(senderName, pos, orient) which provides a client with an updates status of a remote avatar.
    - Server can send bye() to inform all other clients a client has left.

Possible synchronization issues in online games and possible solutions: Due to the nature of network and latency, world sync issues can arise. The solution to this is to perform some sort of data prediction. Example, previous client position history recorded, extrapolate next position by fitting a curve to the points. Move the ghost ahead of time. Issue what if its wrong? Gradually move to actual position to prevent jerkiness.