Introduction to 3D Game Development

Outline

- The computer Game Industry
- Elements of 3D a Game
- The Torque Engine

The computer Game Industry

What is an *indie*? – *An independent game developer.* Not commercial game studios.

- 3D Game Genre and Styles
- Game Platforms
- Game Developer Roles
- Publish Your Game

3D Game Genre and Styles

- Action Games
- Adventure Games
- Role-Playing Games
- Maze and Puzzles Games
- Simulator Games
- Sports Games
- Strategy Games

Action Games

- First-Person Point-of-View (PPOV) or First-Person Shooter (FPS) games
  - *Delta Force* by NovaLogic
  - *Duke Nukem 3D* by 3D Realms
  - *Quake, Wolfenstein* by id Software
  - *Unreal* and its multiplayer version, *Unreal Tournament* by Epic Games
  - *Half-Life series* by vale Software
  - *Counter-Strike* is a modification of Half-Life
  - *Halo series* by Bungie Studio
  - *Perfect Dark* for Nintendo by Rare
  - *TimeSplitters* series by Eidos,
  - *System Shock* by Looking Glass Technology

Doom 3

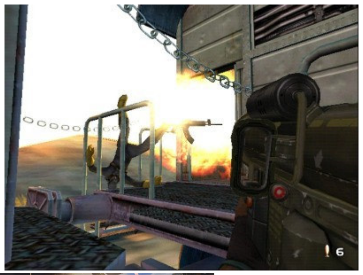
Delta Force

Half-Life 2

Counter-Strike

Unreal

Halo


TimeSplitters

Perfect Dark

System Shock

Action Games

- Third-Person Shooter Games
  - *Grand Theft Auto* series by RockStar North
  - *Red Dead Revolver* by RockStar Sandiego
  - *Gunz* by MAIET Entertainment

Grand Theft Auto

Red Dead Revolver

Gunz

Adventure Games

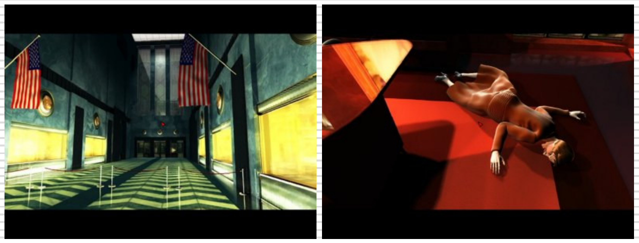
- Adventure games are a type of game, characterized by investigation, that may include
  - exploration,
  - puzzle-solving,
  - interaction with game characters,
  - and have a focus on story telling rather than action challenges.
- Types of adventure games
  - Text-based
  - Graphic
  - Role-Playing games

Adventure Games

- King's Quest Series by Sierra Studios

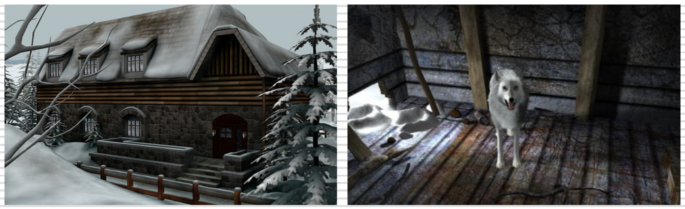
Adventure Games

- The Secrets of Atlantis by Nobilis



Adventure Games

- Nancy Drew: The White Wolf of Icicle Creek by Her Interactive



Role-Playing Games

- A game in which players assume the roles of characters and act out fantastical adventures, the outcomes of which are partially determined by chance, as by the roll of dice.
  - Dungeon Runners by NCsoft
  - GODS: Lands of Infinity SE by Cypron Studios

Role-Playing Games

- Dungeon Runners by NCsoft

Role-Playing Games

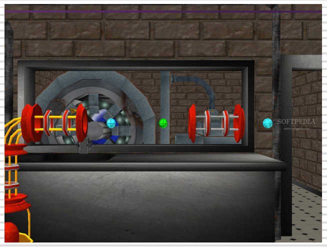
- GODS: Lands of Infinity SE by Cypron Studios

Maze and Puzzle Games

- is a computer game with an emphasis on puzzle solving. The types of puzzles involved can involve logic, strategy, pattern recognition, sequence solving, word completion.
  - Tube Twist Game by Bigfish Games
  - 3D Dragon Maze Game by Gelio Soft

Maze and Puzzle Games

- Tube Twist Game by Bigfish Games



Maze and Puzzle Games

- 3D Dragon Maze Game by Gelio Soft

Simulator Games

- a game that contains a mixture of skill, chance, and strategy to simulate an aspect of reality.
- For example: *MS Flight Simulator*, *SimCity*, *Civilization* and *The Sims*.
- Some simulation games are intended to simulate the real world; others are intended to simulate a fictional world; still others are designed to be able to do both.
  - The Sim 2 by Maxis
  - Black Hawk by Abacus

Simulator Games

- The Sim 2 by Maxis

Okay Tank, what gives? Where did you get these satellite shots of the Nuclear plant? These are classified.

Am I a rich fool? What'd'ya mean "fool"? Sure I'm rich, but I'm no chump. What do you want? Out with it.

Simulator Games

- Black Hawk by Abacus

Sports Games

- a computer or video game that simulates the playing of traditional sports.
- Some games emphasize actually playing the sport (such as the Madden NFL series),
- while others emphasize the strategy behind the sport (such as Championship Manager).
  - Madden NFL 08 by EA Triburon
  - NBA 08 by SCEA

Sports Games

- Madden NFL 08 by EA Triburon

Sports Games

- NBA 08 by SCEA

Strategy Games

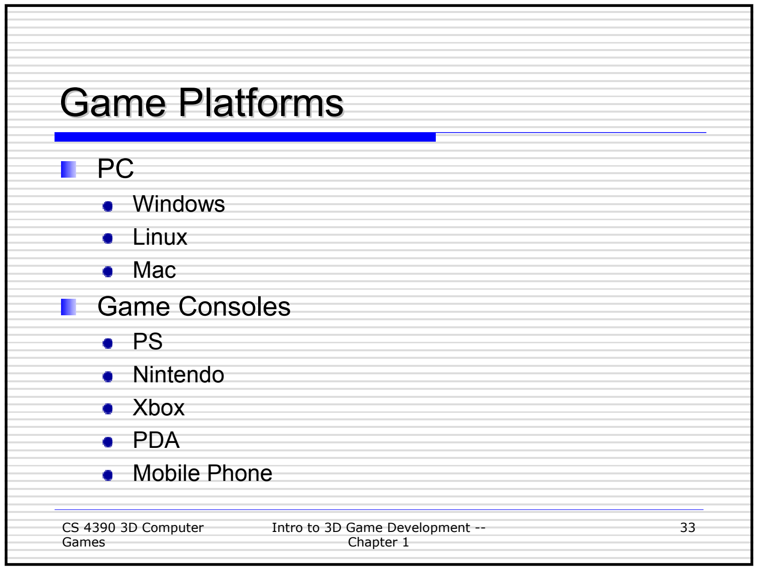
- a video or computer game or other type of game in which the players' decision-making skills have a high significance in determining the outcome.
- Many games include this element to a greater or lesser degree, making demarcation difficult.
  - Kohan II Kings of War by TimeGate Studios
  - American Civil War - The Blue and the Gray by AGEOD

Strategy Games

- Kohan II Kings of War by TimeGate Studios

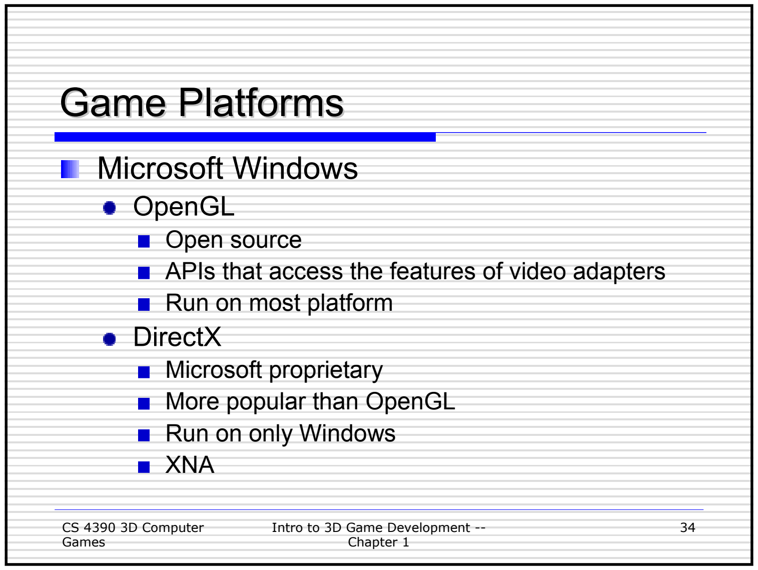
Strategy Games

- American Civil War - The Blue and the Gray by AGEOD



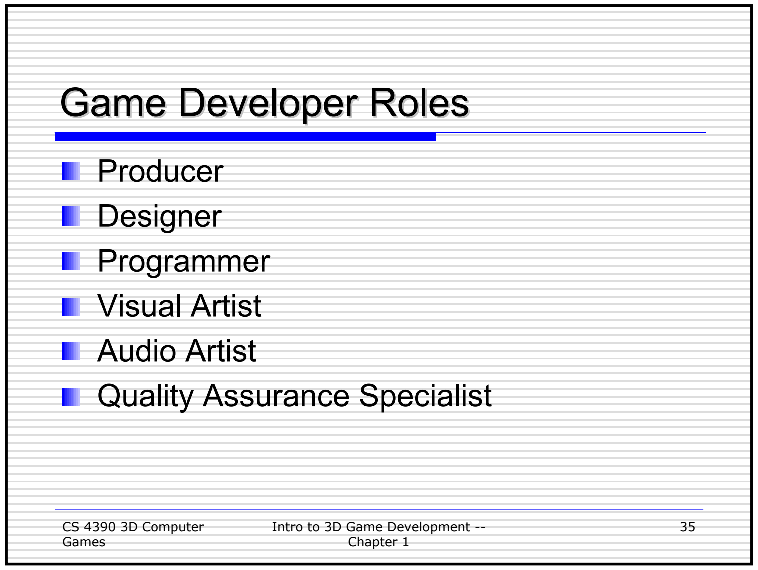
Game Platforms

- PC
  - Windows
  - Linux
  - Mac
- Game Consoles
  - PS
  - Nintendo
  - Xbox
  - PDA
  - Mobile Phone



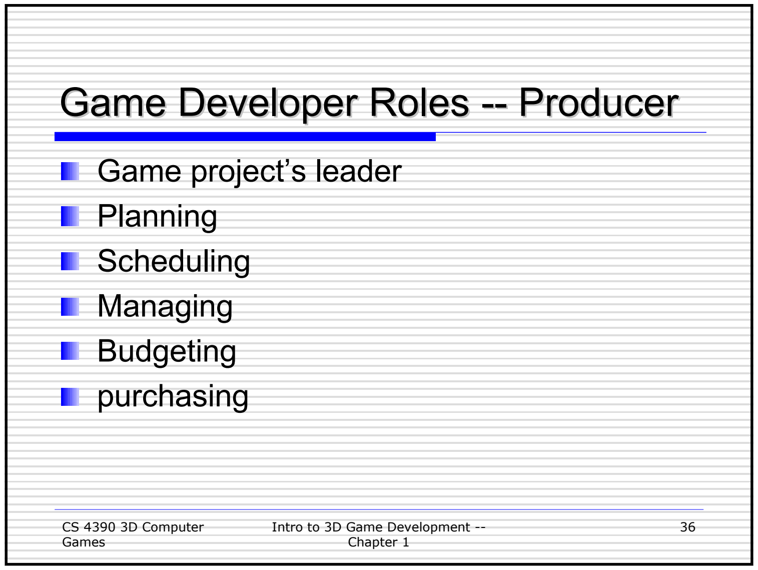
Game Platforms

- Microsoft Windows
  - OpenGL
    - Open source
    - APIs that access the features of video adapters
    - Run on most platform
  - DirectX
    - Microsoft proprietary
    - More popular than OpenGL
    - Run on only Windows
    - XNA



Game Developer Roles

- Producer
- Designer
- Programmer
- Visual Artist
- Audio Artist
- Quality Assurance Specialist



Game Developer Roles -- Producer

- Game project's leader
- Planning
- Scheduling
- Managing
- Budgeting
- purchasing

Game Developer Roles -- Designer

- Game designer -- lead designer, level designer, story-writer designer, model designer, character designer
- Lead designer -- create a plan
  - Maps
  - Games objectives
  - Tools
  - Flow charts
  - Table of characteristics
  - Terrains and Models

Game Developer Roles -- Programmer

- Develop code that makes game come to live
  - Graphic
  - Sound
  - Network
  - Online
  - Interactive

Game Developer Roles -- Visual Artist

- Draw sketches
- Create story board
- 3D Graphics
  - Models
  - Animations
  - Textures

Game Developer Roles -- Audio Artist

- Compose music and sound

Game Developer Roles -- Quality Assurance Specialist

- Extensively test the final game product

Publish Your Game

- Self-publish
- GarageGames
- Known game publishers

Elements of 3D Game

- Game Engine
- Scripts
- GUI
- Models
- Textures
- Sound
- Music
- Support Infrastructure

Elements of 3D Game -- Game Engine

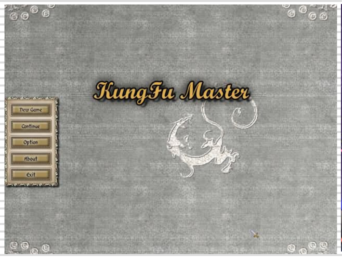
- A game engine is an integrated collection of various computer code objects that together run the video game. These modules include:
  - A graphics module for 2D or 3D
  - A physics module
  - A collision detection module
  - An input/output module
  - A sound module
  - An artificial intelligence module
  - A network module
  - A database module
  - A Graphical User Interface module (GUI)

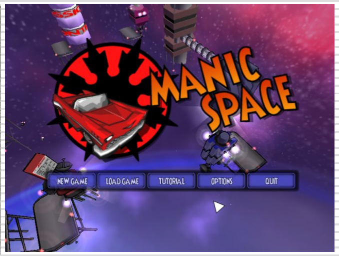
Elements of 3D Game -- Scripts

- The game engine normally provides a scripting language that allows the code to control all elements of the game -- GUI, rendering, Networking, …

Elements of 3D Game -- GUI

- GUI is basically a combination of graphics and scripts that provides visual appearance of the game and accepts the user's control inputs.





Elements of 3D Game -- Models

- 3D models are the most important part of 3D games --
  - Terrain
  - Characters
  - Buildings
  - Trees
  - Vehicles …

Elements of 3D Game -- Textures

- Textures/skins are an important part of rendering the models in 3D scenes.
- They enhance a more realistic appearance.

Elements of 3D Game -- Sound

Elements of 3D Game -- Music

Provides appropriate
- Background sounds
- Event sounds

To fit context and environment of the game

Elements of 3D Game -- Online

- Support Infrastructure
- Web Sites
- Auto-Update
- Support Forums
- Administrative Tools
- Database

The Torque Engine

- Description
- Using Torque
- Installing Torque

Description

- Basic Control Flow
  - Torque uses DemoGame::main() function to initialize libraries and game functions. Then cycles in the main game loop until the program is terminated.
  - The main loop basically calls platform library functions (engine/platform/platform.h) to produce platform events, which then drive the main simulation.
  - Torque handles all the basic event procession functions as follows:
    - Dispatches Windows mouse movements to the GUI),
    - processInputEvent (which processes other input related events)
    - processTimeEvent which computes an elapsed time value based on the time scale setting of the simulation and then:
      - Processes time for server objects (serverProcess() in engine/game/game.cc)
      - Checks for server network packet sends (serverNetProcess() in engine/game/netDispatch.cc)
      - Advances simulation event time (Sim::advanceTime() in engine/console/simManager.cc
      - Processes time for client objects (clientProcess() in engine/game/game.cc)
      - Checks for client network packet sends (clientNetProcess() in engine/gamee/netDispatch.cc)
      - Renders the current frame (GuiCanvas::render() in engine/gui/guiCanvas.cc) c)
      - Checks for network timeouts (dispatchCheckTimeouts() in engine/game/netDispatch.cc)

Description -- Platform Layer

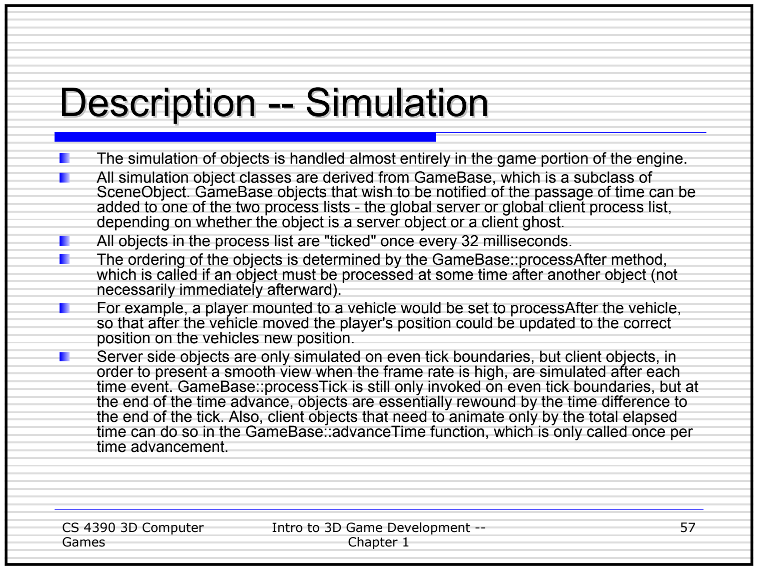
- The platform layer is the foundation of Torque
- Running at the lowest level, it provides a common cross platform, cross architecture interface to the system for the game.
- The platform layer is responsible for handling the details of file and network IO, graphics initialization, device initialization and input, and time event generation
- Standard library calls are proxied through the platform layer, so that the game code can be safe from platform specific idiosyncrasies

Description -- Console

- The console module, rooted in engine/console/console.h, is a combined compiler and interpreter runtime that serves as the foundation for Torque applications.
- All GUIs, game objects, interfaces, and game logic are handled through the console.
- The language itself is syntactically similar to a typeless C++, with some additional features that allow for easier mod development.
- Console scripts can be loaded via the exec() console command from the console window (brought up using the ~ key) or they can be loaded automatically from a mod via that mod's main.cs.

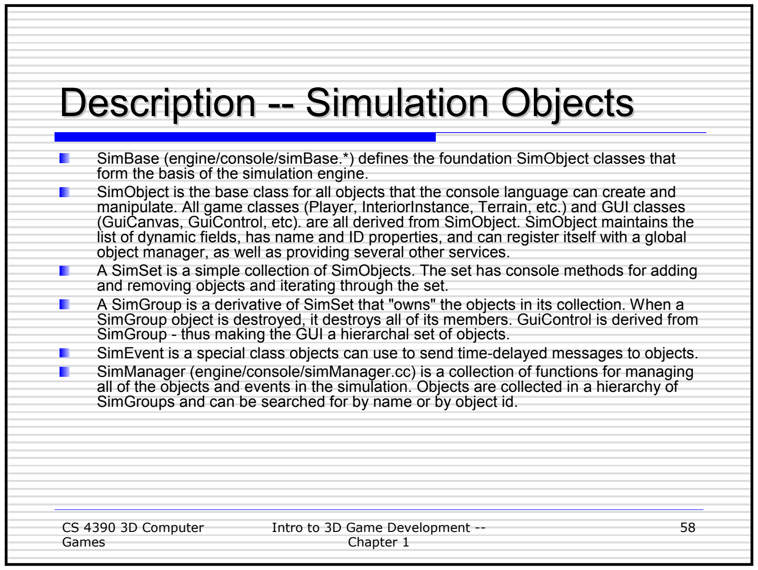
Description -- Input Model

- Input events come from the OS, are translated in the platform layer and then posted to the game.
- By default the game then checks the input event against a global action map (which supercedes all other action handlers). If there is no action specified for the event, it is passed on to the GUI system. If the GUI does not handle the input event it is passed to the currently active (non-global) action map stack.
- Platform Input --
  - Platform specific code translates OS-specific events into uniform Torque input events.
  - These events are posted into the main application event queue via a call to GameInterface::processEvent() (remember, Game points to a subclass of GameInterface).
  - The default behavior for the GameInterface class is to pass all input events to GameInterface::processInputEvent(), which in the example DemoGame calls ActionMap::handleEventGlobal, followed by Canvas->processInputEvent (if not handled by the global map), and if neither of those handles it, passes it to ActionMap::handleEvent.
- Action Maps --
  - Action maps map platform input events to console commands.
  - Any platform input event can be bound in a single generic way - so in theory the game doesn't need to know if the event came from the keyboard, mouse, joystick or some other input device.
  - This allows users of the game to map keys and actions according to their own preferences.
  - There is one defined ActionMap object that is processed first for all events called GlobalActionMap.
  - Game action maps are arranged in a stack for processing - so individual parts of the game can define specific actions - for example when the player jumps into a vehicle it could push a vehicle action map and pop the default player action map.



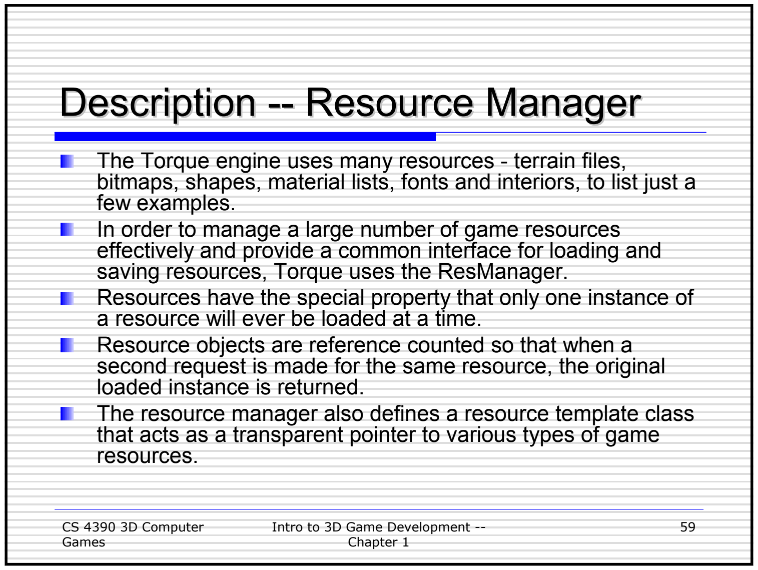
Description -- Simulation

- The simulation of objects is handled almost entirely in the game portion of the engine.
- All simulation object classes are derived from GameBase, which is a subclass of SceneObject. GameBase objects that wish to be notified of the passage of time can be added to one of the two process lists - the global server or global client process list, depending on whether the object is a server object or a client ghost.
- All objects in the process list are "ticked" once every 32 milliseconds.
- The ordering of the objects is determined by the GameBase::processAfter method, which is called if an object must be processed at some time after another object (not necessarily immediately afterward).
- For example, a player mounted to a vehicle would be set to processAfter the vehicle, so that after the vehicle moved the player's position could be updated to the correct position on the vehicles new position.
- Server side objects are only simulated on even tick boundaries, but client objects, in order to present a smooth view when the frame rate is high, are simulated after each time event. GameBase::processTick is still only invoked on even tick boundaries, but at the end of the time advance, objects are essentially rewound by the time difference to the end of the tick. Also, client objects that need to animate only by the total elapsed time can do so in the GameBase::advanceTime function, which is only called once per time advancement.



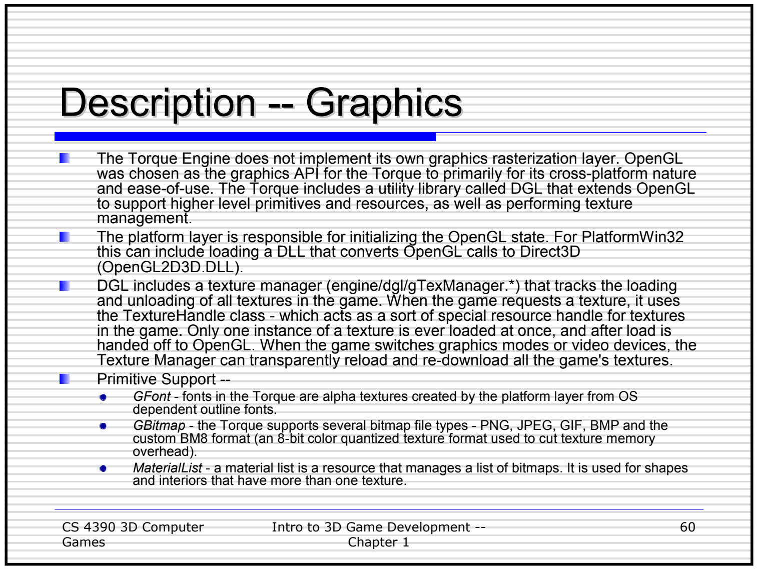
Description -- Simulation Objects

- SimBase (engine/console/simBase.*) defines the foundation SimObject classes that form the basis of the simulation engine.
- SimObject is the base class for all objects that the console language can create and manipulate. All game classes (Player, InteriorInstance, Terrain, etc.) and GUI classes (GuiCanvas, GuiControl, etc). are all derived from SimObject. SimObject maintains the list of dynamic fields, has name and ID properties, and can register itself with a global object manager, as well as providing several other services.
- A SimSet is a simple collection of SimObjects. The set has console methods for adding and removing objects and iterating through the set.
- A SimGroup is a derivative of SimSet that "owns" the objects in its collection. When a SimGroup object is destroyed, it destroys all of its members. GuiControl is derived from SimGroup - thus making the GUI a hierarchal set of objects.
- SimEvent is a special class objects can use to send time-delayed messages to objects.
- SimManager (engine/console/simManager.cc) is a collection of functions for managing all of the objects and events in the simulation. Objects are collected in a hierarchy of SimGroups and can be searched for by name or by object id.



Description -- Resource Manager

- The Torque engine uses many resources - terrain files, bitmaps, shapes, material lists, fonts and interiors, to list just a few examples.
- In order to manage a large number of game resources effectively and provide a common interface for loading and saving resources, Torque uses the ResManager.
- Resources have the special property that only one instance of a resource will ever be loaded at a time.
- Resource objects are reference counted so that when a second request is made for the same resource, the original loaded instance is returned.
- The resource manager also defines a resource template class that acts as a transparent pointer to various types of game resources.



Description -- Graphics

- The Torque Engine does not implement its own graphics rasterization layer. OpenGL was chosen as the graphics API for the Torque to primarily for its cross-platform nature and ease-of-use. The Torque includes a utility library called DGL that extends OpenGL to support higher level primitives and resources, as well as performing texture management.
- The platform layer is responsible for initializing the OpenGL state. For PlatformWin32 this can include loading a DLL that converts OpenGL calls to Direct3D (OpenGL2D3D.DLL).
- DGL includes a texture manager (engine/dgl/gTexManager.*) that tracks the loading and unloading of all textures in the game. When the game requests a texture, it uses the TextureHandle class - which acts as a sort of special resource handle for textures in the game. Only one instance of a texture is ever loaded at once, and after load is handed off to OpenGL. When the game switches graphics modes or video devices, the Texture Manager can transparently reload and re-download all the game's textures.
- Primitive Support --
  - *GFont* - fonts in the Torque are alpha textures created by the platform layer from OS dependent outline fonts.
  - *GBitmap* - the Torque supports several bitmap file types - PNG, JPEG, GIF, BMP and the custom BM8 format (an 8-bit color quantized texture format used to cut texture memory overhead).
  - *MaterialList* - a material list is a resource that manages a list of bitmaps. It is used for shapes and interiors that have more than one texture.

Description -- Graphics

- Primitive Rendering --
  - The DGL support functions support a wide variety of common 2D rendering primitives. Bitmaps (loaded as textures) can be rendered via the dglDrawBitmap(), dglDrawBitmapStretch(), dglDrawBitmapSR() (sub-region), and dglDrawBitmapStretchSR() functions. Text can be rendered using dglDrawText() and dglDrawTextN(). Dgl also supports drawing of lines, rectangles and filled rectangles.
  - Unlike default OpenGL, the screen coordinate space set up for 2D rendering in the Torque is a traditional 2D scheme where (0,0) is in the upper left corner of the screen and +Y goes down the screen. This requires (in the case of 3D) calling dglSetViewport rather than glViewport.
  - For 2D rendering, DGL viewport management is simplified by dglSetClipRect().
- 3D Redering --
  - There are several key differences in how the Torque does rendering from the default OpenGL. First, the coordinate system is set up to look down the +Y axis instead of -Z. This means dgl replaces the call to glFrustum() with a call to dglSetFrustum(). Also, all Torque matrices are organized in standard C array form - with the second element in the array corresponding to the first row, second column. This is the opposite of OpenGL, so DGL supplies alternates to glLoadMatrix() and glMultMatrix(), appropriately named dglLoadMatrix() and dglMultMatrix() respectively.
  - 3D points can be converted to 2D screen points using the dglPointToScreen() function, and a measure of projected screen size of an object can be determined using the dglProjectRadius() function.

Description -- 3D Rendering

- Torque has a modular, extensible 3D world rendering system. Subclasses of the GuiTSCtrl override the GuiTSCtrl::processCameraQuery() and GuiTSCtrl::renderWorld() methods to define the camera orientation/FOV, and draw the 3D scene using OpenGL drawing commands respectively.
- GuiTSCtrl manages setting up the viewport, modelview matrix and projection matrix.
- The Torque example code GameTSCtrl class calls the global functions GameProcessCameraQuery() and GameRenderWorld().
- GameProcessCameraQuery() returns the viewing camera of the current control object (the object in the simulation that the player is currently controlling), then GameRenderWorld makes the client scene graph object render the world.

Description -- 3D Rendering

- Scene Graphs --
  - The scene graph library (engine/sceneGraph) is, on the client, responsible for traversing the world scene and determining which objects in the world should be rendered given the current camera position, and on the server, determines what objects should be sent to each client based on that client's position in the world.
  - The world in the SceneGraph is divided into zones - volumes of space bounded by solid areas and portals. The outside world is a single zone, while interior objects can have multiple interior zones. SceneGraph::findZone() finds the zone of a given 3D point and reports which SceneObject owns that zone. SceneGraph::rezoneObject() determines which zone or zones contain a SceneObject instance. At render time, the scene is traversed starting from the zone that contains the camera, clipping each zone's objects to the visible portal set from the zones before it. Scoping of network objects is performed in SceneGraph::scopeScene().
  - The scene graph traversal is complicated by transform portals. Transform portals are objects like mirrors or teleporters through which the world can be viewed using a different transform than the normal camera transform. When SceneGraph::buildSceneTree() encounters an object with a transform portal, it constructs a new SceneState object for rendering that portal's contents.
  - Every renderable world object in the scene derives from the SceneObject base class. As the world is traversed, visible objects are asked to prepare one or more SceneRenderImage objects (in SceneObject::prepRenderImage()) that are then inserted into the current SceneState via SceneState::insertRenderImage(). Render images are sorted based on translucency and rendered from SceneObject::renderObject(). This system allows, for example, an interior object with multiple translucent windows to render the building first, followed by other objects, followed by the building's windows. Objects can insert any number of images for rendering.

Description -- Terrain

- The terrain library (engine/terrain) is the home for objects that render the outside world, including instances of the Sky, TerrainBlock and WaterBlock classes.
- The Sky object renders the outside sky and cloud layers and maintains the visible distance and fog distance settings for the world. The sky also tracks vertical fog layers and installs them into the SceneGraph for rendering.
- TerrainBlock manages a single 256x256 infinitely repeating block of heightfield terrain.
- Terrain heightfield data is stored and loaded using the TerrainFile resource class (Resource<TerrainFile>) so that a single terrain data file can be shared between server and client, when both are on the same execution instance.
- The TerrainRender static class is used by TerrainBlock instances for rendering. The TerrainRender::renderBlock() function renders the current repeating block of terrain.
- There exists a TerrainManager resource which allows you to have NxM repeating areas of terrain, increasing the non-repeating terrain area.
- The terrain is textured by software blending base material textures into new material textures and then mapping those across 16 or more terrain squares based on the distance from the square. Blender performs the blending of terrain textures and includes a MMX assembly version to speed the process (x86 architectures only).
- The WaterBlock class manages a single block of water, which may or may not be infinitely repeating. Water is dynamically detailed based on distance, so nearby water is more highly tessellated. Though the surface of a water block is rectangular, the actual coverage of the water area can be set to seed fill from a point on the surface, allowing the water to fill a mountain crater, for example, without leaking outside the corner edges.

Description -- Interior

- The interior library (engine/interior) manages the rendering, collision and IO for interior objects. The InteriorInstance SceneObject class manages a single interior. The InteriorResource class manages the data associated with one definition of an interior, multiple instances of which may exist at any one time. Interiors manage zones for the scene graph, and may have subobjects that, for example, render a mirrored view (MirrorSubObject). The InteriorLMManager class manages lightmaps for all currently loaded interiors - sharing lightmaps among instances where possible.
- Interiors are converted to DIF by the tool Map2DIF (formerly known as Morian). The source files are just Quake-style .map files - lists of convex physical "brushes" that define the solid areas of the interior. Special brushes are used to define zone portal boundaries and objects such as doors and platforms.

Description -- 3Space

- The 3Space library (engine/ts) manages the display and animation of shape models in the world. The 3Space shape resource class TSShape can be shared between multiple TSShapeInstance instances.
- The TSShape class manages all the static data for a shape - mesh data, animation keyframes, material lists, decal information, triggers and detail levels (for dynamically detailed shapes).
- The TSShapeInstance class manages animation, rendering and detail selection for an instance of a shape.
- The TSShapeInstance class uses the TSThread class to manage one of the concurrently running animations on an instance. TSShapeInstance::addThread() initializes a new thread on a shape instance, and TSShapeInstance::setSequence() sets an animation sequence for a given thread. Each thread can be individually advanced in time, or can be set on a time scale that is used when all threads are advanced in TSShapeInstance::advanceTime(). A thread can also manage transitions between sequences with TSShapeInstance::transitionToSequence().
- TSShape animation sequences can be composed of node/bone animation (for example, joints in an explosion), material animation (a texture animation on an explosion) and mesh animation (a morphing blob - note most mesh animations can be accomplished with node scale and rotation animations). Animations can also contain visibility tracks so that some meshes in the shape are not visible until an animation is played.

Description -- Networking

- Torque was designed from the foundations up to offer robust client/server networked simulations. Performance over the internet drove the design for the networking model.
- Torque attempts to deal with three fundamental problems of network simulation programming - limited bandwidth, packet loss and latency.
- For a more detailed, if somewhat outdated, description of the Torque network architecture, see "The Tribes II Engine Networking Model" paper by Tim Gift and Mark Frohnmayer and the accompanying PowerPoint slides in the Torque documentation area on GarageGames.com.
- An instance of Torque can be set up as a dedicated server, a client, or both a client and a server. If the game is a client AND a server, it still behaves as a client connected to a server - instead of using the network, however, the NetConnection object has a short-circuit link to another NetConnection object in the same application instance.
- Bandwidth is a problem because in the large, open environments that Torque allows, and with the large number of clients that Torque supports (up to 128 per server, or beyond, if the network/server can handle it), potentially many different objects can be moving and updating at once.
- The Torque uses three main strategies to maximize available bandwidth.
  - First, it prioritizes data, sending updates to what is most "important" to a client at a greater frequency than it updates data that is less important.
  - Second, it sends only data that is necessary - using the BitStream class, only the absolute minimum number of bits needed for a given piece of data will be sent. Also, when object state changes, Torque only sends the part of the object state that changed.
  - Last, Torque caches common strings (NetStringTable) and data (SimDataBlock) so that they only need to be transmitted once.

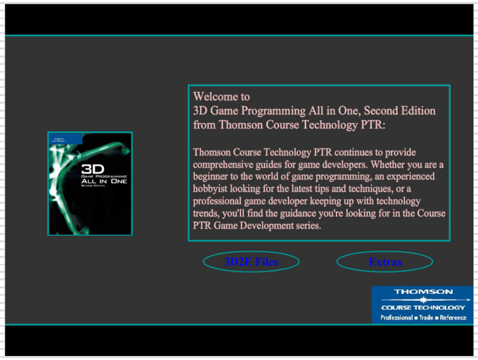
Description -- Networking

- Packet loss is a problem because the information in lost data packets must somehow be retransmitted, yet in many cases the data in the dropped packet, if resent directly, will be stale by the time it gets to the client.
- For example,
  - suppose that packet 1 contains a position update for a player and packet 2 contains a more recent position update for that same player.
  - If packet 1 is dropped but packet 2 makes it across the engine shouldn't resend the data that was in packet 1 - it is older than the version that was received by the client. In order to minimize data that gets resent unnecessarily, the engine classifies data into four groups:
- - Unguaranteed Data (NetEvent) - if this data is lost, don't re-transmit it. An example of this type of data could be real-time voice traffic - by the time it is resent subsequent voice segments will already have played.
- - Guaranteed Data (NetEvent) - if this data is lost, resend it. Chat messages, messages for players joining and leaving the game and mission end messages are all examples of guaranteed data.
- - Most-Recent State Data (NetObject) - Only the most current version of the data is important - if an update is lost, send the current state, unless it has been sent already. - Guaranteed Quickest Data (Move) - critical data that must get through as soon as possible.

Description -- Networking

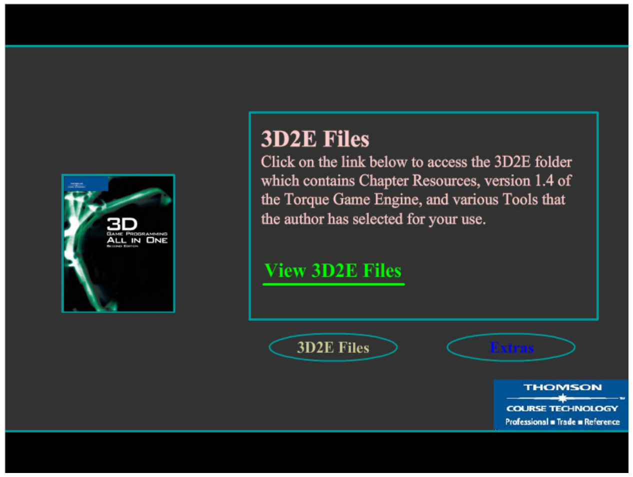
- Latency is a problem in the simulation because the network delay in information transfer (which, for modems, can be up to a quarter of a second or more) makes the client's view of the world perpetually out-of-sync with the server.
- Twitch FPS games, for which Torque was initially designed, require instant control response in order to feel anything but sluggish.
- Also, fast moving objects can be difficult for highly latent players to hit. In order to solve these problems Torque employs several strategies:
  - ***Interpolation*** is used to smoothly move an object from where the client thinks it is to where the server says it is.
  - ***Extrapolation*** is used to guess where the object is going based on its state and rules of movement.
  - ***Prediction*** is used to form an educated guess about where an object is going based on rules of movement and client input.
- The network architecture is layered: at the bottom is the platform layer, above that the notify protocol layer, followed by the `NetConnection` object and event management layer.

Installing Torque



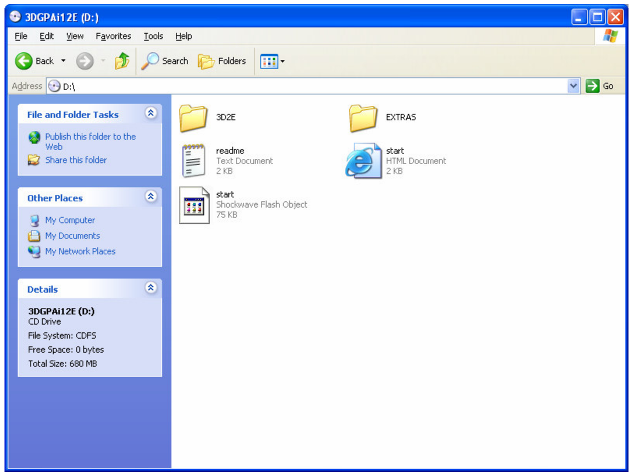
Welcome to
3D Game Programming All in One, Second Edition
from Thomson Course Technology PTR:

Thomson Course Technology PTR continues to provide comprehensive guides for game developers. Whether you are a beginner to the world of game programming, an experienced hobbyist looking for the latest tips and techniques, or a professional game developer keeping up with technology trends, you'll find the guidance you're looking for in the Course PTR Game Development series.



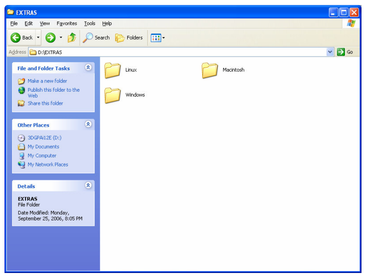
3D2E Files
Click on the link below to access the 3D2E folder which contains Chapter Resources, version 1.4 of the Torque Game Engine, and various Tools that the author has selected for your use.

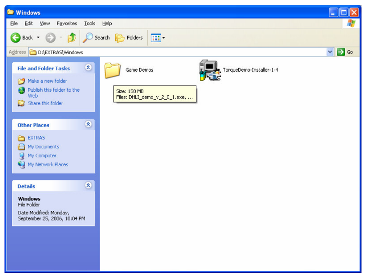
View 3D2E Files

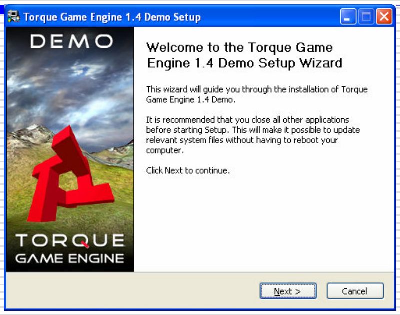


3DGPAi12E (D:)
CD Drive
File System: CDFS
Free Space: 0 bytes
Total Size: 680 MB

3D2E
EXTRAS
readme — Text Document — 2 KB
start — HTML Document — 2 KB
start — Shockwave Flash Object — 75 KB





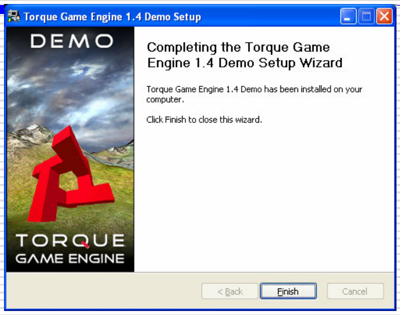


Welcome to the Torque Game Engine 1.4 Demo Setup Wizard

This wizard will guide you through the installation of Torque Game Engine 1.4 Demo.

It is recommended that you close all other applications before starting Setup. This will make it possible to update relevant system files without having to reboot your computer.

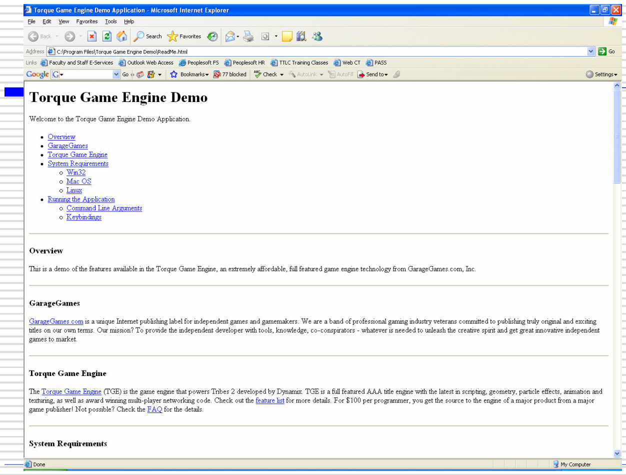
Click Next to continue.



Completing the Torque Game Engine 1.4 Demo Setup Wizard

Torque Game Engine 1.4 Demo has been installed on your computer.

Click Finish to close this wizard.


Torque Game Engine Demo Application - Microsoft Internet Explorer
Address C:\Program Files\Torque Game Engine Demo\ReadMe.html
Links Faculty and Staff E-Services Outlook Web Access Peoplesoft FS Peoplesoft HR TTLC Training Classes Web CT PASS
Done
My Computer

Torque Game Engine Demo

Welcome to the Torque Game Engine Demo Application.

- Overview
- GarageGames
- Torque Game Engine
- System Requirements
  - Win32
  - Mac OS
  - Linux
- Running the Application
  - Command Line Arguments
  - Keybindings

Overview

This is a demo of the features available in the Torque Game Engine, an extremely affordable, full featured game engine technology from GarageGames.com, Inc.

GarageGames

GarageGames.com is a unique Internet publishing label for independent games and gamemakers. We are a band of professional gaming industry veterans committed to publishing truly original and exciting titles on our own terms. Our mission? To provide the independent developer with tools, knowledge, co-conspirators - whatever is needed to unleash the creative spirit and get great innovative independent games to market.

Torque Game Engine

The Torque Game Engine (TGE) is the game engine that powers Tribes 2 developed by Dynamix. TGE is a full featured AAA title engine with the latest in scripting, geometry, particle effects, animation and texturing, as well as award winning multi-player networking code. Check out the feature list for more details. For $100 per programmer, you get the source to the engine of a major product from a major game publisher! Not possible? Check the FAQ for the details.

System Requirements