2025

Technical Design Document

GDV5001, WRIT1

HANCOX, LIAM

Contents

Introduction

This document will investigate the design and implementation of various features used in the creation of a clone version of Bullfrog's *Dungeon Keeper 2* title (Bullfrog, 1999). For each of the key elements, research and analysis has been carried out to understand or make informed assumptions about how the original title has achieved them.

3D Model Rendering

The Dungeon Keeper 2 title used the Graphics API Direct3D (PCGamingWiki, 2024) as they wanted to make a 3D game that utilised hardware acceleration. The lead of the development team is quoted saying *"We developed a custom engine that could render the entire dungeon environment and its inhabitants in 3D, which meant bringing in a much larger team—3D modelers, riggers, animators, the works,"*,(Lane, 2024) this upgrade was introduced to replace the original titles pre-rendered sprites with polygonal models allowing the game world and its inhabitants to be a 3D (PCZone, 1999). The OpenGL graphics API (KhronosGroup, 2025) was used to achieve this, a collection of OBJ files were loaded through a manifest via a model factory, where an ASSIMP (Open Asset Import Library) library parses the file and extracts all the relevant information such as vertex positions and normal etc and this data is then uploaded to OpenGL via vertex buffer objects (VBOs), vertex array objects (VAOs) and an index buffer to improve render speed. Once all objects are loaded a render function is called which iterates over all the game objects that have been loaded from the manifest and draws the objects to screen.

Texture Mapping

Dungeon Keeper 2 makes use of many textures, including varying textures across walls and terrain. Due to a lack of information on how Dungeon Keeper 2 handles its texturing an assumption is made from watching footage of gameplay that each terrain/wall type has its associated predefined group of textures and during generation the game randomly selects from the appropriate group for each instance to avoid repetition. There was also the addition of bump mapping (Autodesk, 2024), a texture mapping technique, in the 1.7 update for the game, this added extra visual effects to the water and lava (Wiki, 2024). For this clone, when each ExampleGO object is loaded it reads its associated texture from the manifest and stores it, then during initialisation it accesses Scenes texture manager and gets the appropriate OpenGL texture ID (Gluint) where during the PreRender method after allowing the object to receive 2D textures it also binds the stored texture to the retrieved texture unit, in this case 0 (GL_TEXTURE0) is the main texture and 1 (GL_TEXTURE1) is for normal maps if a normal map is required.

Render is then called where the objects model is drawn and the selected textures applied.

Lighting

From watching footage of gameplay of Dungeon Keeper 2 (SeanyC, 2020), it is observed that the types of lights used seem to mostly consist of wall mounted torches that are point lights as well as one that follows the players cursor, with a directional light cast over the entire map. Within the modding communities at Keeper Klan (KeeperKlan, 2025) and Reddit (Reddit, 2025), it has been speculated that Bullfrog calculated these point lights per-vertex via Gouraud shading (GeeksforGeeks, 2023), where the calculation of the lights direction and the surface normals around the vertex are used to blend the results across the surface of a 3D object to give a smooth but simple appearance dependant of the amount of vertices available on a surface or model. The lighting for this clone application also makes use of point lights and primarily through the derived class TorchLight, which inherits from the PointLight class. The TorchLight class was created as they were required to act and look the same, these are initialised, and its attributes are loaded from the manifest and a shared global vector is created to store all this information of the number of active lights needed and to pass this information to the BuildDungeon class. This class differs by making use of the Tick method where it uses the shared_ptr of the vector and loops through all instances of the torchlight and applies a random range between clamped red and green colour values to give a flickering effect of the torchlight. SetRenderValues method is then called which passes the information/attributes to the shader, where the shader calculates all the light information passed in from the structs, the vert shader gives the per-fragment data such as world position, tangent, bi-tangent and normals needed for lighting and normal mapping, this is then passed to the frag shader to apply those normal results to surfaces to calculate how the light interacts with the surface.

Transparency

As no information could be found directly about how Dungeon Keeper 2 rendered transparent objects, modding forums, gameplay footage of the original game and research into the techniques around the time of the games release were looked into and an educated guess has been made. The user “SuperFriendBFG” got the GoG.com community discussions forum (GoG.com, 2011) is seen posting *“…Dungeon Keeper 2 uses 16-bit Z-Buffering. It's an old rendering method that is no longer used in modern games….”* , this Z-Buffering technique ensures that closer 3D objects appear in-front of farther ones. As each pixel of a 3D object is rendered to the screen, its distance from the camera (depth value) is compared against any object in that position on screen and

checks if that object is closer to the cameras position, if so, it will replace the information in the buffer with the new value (Wiki, 2025). This clone also makes use of this technique to ensure that the opaque objects follow this rule, and all the opaque objects render correctly. It also makes use of the Transparency Sorting technique in which all of the opaque objects are rendered first, then all of the transparent objects are collected into a vector and sorted from the furthest to the closest from the camera and rendered in that order (KhronosGroup, 2015). It also takes advantage of alpha blending (PCMag, 2025), where OpenGL allows the combination of the alpha channel with the colours of the image to make its transparent.

Cameras

During the main gameplay in Dungeon Keeper 2, the game uses an orthographic camera which is set a fixed distance above the dungeon. This camera can rotate horizontally around the Y-axis. Also, the possession spell (Wiki, 2025) gives the player access to a first-person camera, this temporarily changes to the camera letting the player see and interact with the game world though the eyes of a minion/creature. This application has a base class Camera, this has the base view and projection matrix generation, ArcballCamera, FPCamera and OrthoCamera all inherit from the camera class, but each type has different output of how they are rendering the scene. The Orbit camera (ArcballCamera) has a fixed point (lookAt) that the camera can rotate around using spherical coordinates and is used here to quickly inspect the rendered game world. The first-person camera (FPCamera) put the player into the game world and is the first pass towards the creature possession that is in the original game this is a clone of. The Orthographic camera implemented (OrthoCamera) is used as the main display of the rendered game world and gives a higher wider view over the level.

Interaction

Dungeon Keeper 2 has a plethora of available interactions to the player, including but not limited to Dungeon construction - where you can build rooms inside your dungeon, Creature Management – Assign tasks, send them to attack, slap, pick up and drop them and the ability to move around the game world (Bullfrog, 1999a). The movement is the main interaction focused on in this application, and each camera moves differently but do not the key presses themselves, the key presses are stored and handled in the main.cpp in a struct that sets the appropriate Boolean, that then gets and sets the correct movement direction method. In Scene, dependant on which camera is currently active handles which classes Move() method gets called. Within the Move() method of the FPCamera, the forward vector is calculated as the direction the camera is facing, and the right vector is calculated from the cross product of the forward vector and the

world up vector. The OrthoCamera works out the same logic to work out the forward and right vectors but instead travels across the map due to a locked theta value.

Other Aspects

Wandering Monsters

As seen in Dungeon Keeper 2, the characters/creatures use AI to navigate around the level and according to the Manual each creature has different behaviours and strategies (Bullfrog, 1999). Due to all of these varying movements and actions it is likely that it makes use of an FSM (Finite State Machine) which is a common AI architecture used to control game entities giving them the illusion of intelligence. These will assign/set states depending on various variables for example, if a creature has nothing to do it will be set to an Idle state, then it if is set to Patrol it will walk around and if it comes close enough to an enemy it may enter a Chase state. In this clone, moving creatures are created in the Wandering class which inherits from ExampleGO. These objects travel through the dungeon until they reach the centre of each of the square dungeon rooms, these positions have been hard coded and put into an array of vec3's, where when they reach any of these positions the turnPlace() method is called where it checks which one of the 9 positions it is in and randomly selects from its available directions and moves in that direction and repeats indefinitely. This was added to give the illusion that the creatures are wandering around the dungeon and also to stop them just leaving the map.

Manifest

The manifest is a .txt file that is used to declare elements in the project used for the scene construction (Cameras, Lights, Models, Textures, Shaders and Game Objects). These sections hold all of the information needed to create each element, when the application starts all of this information get parsed and uses Factories to create each element. The manifest will know specific base data such as the objects type, name and position etc that will be sent to these factories where they will be created and within their own classes specific logic will be given. This makes it simple to add extra models, textures and game object types in a readable format where the developer needs only to update the manifest and the appropriate factory.

Map generation

The BuildDungeon class is responsible for map generation and makes use of the predetermined rooms setup within the manifest. Each part of the dungeon works with an origin system, where using the DUNGEONMAIN game object as an example, it receives a predefined number of origins (NOORIGINS), in this case 9 and places them into a vec3 list, and a predefined number of game objects (NOOBJ), is this case 28 and places these into a vec3 list, and it goes to each hardcoded origin and places the 28 walls into the hardcoded positions and creates the shape of the room, this then gets pushed back to a finalObjLocation and this gets rendered. This is repeated for all the other rooms, corridors and torches. The difference within the Torch is that for each of the final locations of the torch it also calls to the TorchLight class and places a Torchlight at that location with an offset to give the appearance of the torch actually giving off light. The way this is setup, means that every time a Torch game object is added to the map it will automatically receive the appropriate Torchlight Light.

Referencing

Autodesk (2024) *Bump mapping software: What is bump mapping?*, *Autodesk*. Available at: https://www.autodesk.com/solutions/bump-mapping (Accessed: 08 May 2025).

Brooker, C. (1999) 'Lock up your daughters, here comes... Dungeon Keeper 2'. *PC Zone*, no. 77, June, pp. 44–49. ISSN 0967-8220. (Accessed: 08 May 2025)

Bullfrog (1999) Dungeon Keeper 2 [Video game]. Bullfrog Productions.

Bullfrog (1999a) *Dungeon-keeper-2-manual.PDF*. Available at: https://retrogamer.biz/wp-content/uploads/2016/06/Dungeon-Keeper-2-Manual.pdf (Accessed: 13 May 2025).

GeeksforGeeks (2023) *Gouraud shading in Computer Graphics*, *GeeksforGeeks*. Available at: https://www.geeksforgeeks.org/gouraud-shading-in-computer-graphics/ (Accessed: 13 May 2025).

GoG.com (2011) *Dungeon Keeper 2 Hardware Acceleration?*, *Dungeon keeper 2 hardware acceleration? , page 1 - forum - gog.com*. Available at: https://www.gog.com/forum/dungeon_keeper_series/dungeon_keeper_2_hardware_acceleration (Accessed: 13 May 2025).

KeeperKlan (2025) *Keeper klan forum*. Available at: https://keeperklan.com/ (Accessed: 13 May 2025).

KhronosGroup (2025) *The industry's foundation for High Performance Graphics*, *OpenGL.org*. Available at: https://www.opengl.org/ (Accessed: 11 May 2025).

KhronosGroup (2015) *Transparency sorting*, *Transparency Sorting - OpenGL Wiki*. Available at: https://www.khronos.org/opengl/wiki/transparency_Sorting (Accessed: 13 May 2025).

Lane, R. (2024) *Hexes, drugs, rock and trolls: The rise and fall of dungeon keeper - epic games store*. Available at: https://store.epicgames.com/en-US/news/dungeon-keeper-rise-and-fall-interview-peter-molyneux (Accessed: 12 May 2025).

PCGamingWiki (2024) *Dungeon keeper 2*, *Dungeon Keeper 2 - PCGamingWiki PCGW - bugs, fixes, crashes, mods, guides and improvements for every PC game*. Available at: https://www.pcgamingwiki.com/wiki/Dungeon_Keeper_2#API (Accessed: 08 May 2025).

PCMag (2025) *Definition of alpha blending*, *PCMAG*. Available at: https://www.pcmag.com/encyclopedia/term/alpha-blending (Accessed: 13 May 2025).

Reddit (2025) *R/gamedev - fake lights and shadows in Dungeon Keeper 2*. Available at: https://www.reddit.com/r/gamedev/comments/hrz910/fake_lights_and_shadows_in_dungeon_keeper_2/ (Accessed: 13 May 2025).

SeanyC (2020) *YouTube*. Available at: https://www.youtube.com/watch?v=E5leriZUYF8&list=PLNbVimT9PLw3Fzi0zOcWh8y2CIIBMfajp&index=4 (Accessed: 13 May 2025).

Wiki, C. to D.K. (2024) *Bump mapping*, *Dungeon Keeper Wiki*. Available at: https://dungeonkeeper.fandom.com/wiki/Bump_Mapping (Accessed: 12 May 2025).

Wiki, C. to D.K. (2025) *Dungeon keeper 2*, *Dungeon Keeper Wiki*. Available at: https://dungeonkeeper.fandom.com/wiki/Dungeon_Keeper_2 (Accessed: 12 May 2025).

Wiki (2025) *Z-buffering*, *Wikipedia*. Available at: https://en.wikipedia.org/wiki/Z-buffering (Accessed: 13 May 2025).