DirectX 11.2 in Visual Studio 15 [closed]

Asked 7 years, 10 months ago Modified 7 years, 10 months ago

Viewed 1k times

0

Closed. This question needs details or clarity. It is not currently accepting answers.

Want to improve this question? Add details and clarify the problem by editing this post.

Closed 7 years ago.

Improve this question

My friends and I go to a vocational school for computer programming. We got together and decided to start making games, but the problem is only two of us have. I've made a text adventure game using lots of buttons and lots of forms in Visual Studio, and my other friend has made a few games using GameMaker. All of us have the same basic knowledge of Visual Basic (due to being in the same Programming course) and we want to try our hands at making a 3D game. I've done a little bit of research, and determined that we may need to use DirectX if we wish to keep using Visual Studio.

The problem is, I don't know how to start really. I've tried looking it up. Is it possible to access the API using just Windows and Visual Studio?

Also, based on what I can tell, we need to know C++. If that's true, then we'll convert to C++ to make games. It shouldn't be too difficult, I'd imagine.

visual-studio directx-11

Share

Improve this question

Follow

asked Feb 3, 2017 at 1:50

Ellie905

11 ● 2

A 3D Game is more than just about 3D Graphics. 3D Graphics has thousands of topics and discussions out there about the various algorithms and techniques to perform all the various different things that need or could be done with a 3D Graphic. The Graphics is just 1 of Many Parts to a working 3D Game. – Francis Cugler Feb 3, 2017 at 7:21

2 Answers

Sorted by: Highest score (default)

4

Planning on making a game in 3D? This is no easy task and is very daunting. This isn't to scare you away from pursuing it, but to let you know what is down the road ahead of you. I'll break this down into several lists to give you a clear or better understanding.

Things to Consider

- *What type of game are we going to make?*
    - 2D Side Scroll, 2D Bitmaps and Sprites, Partial 3D or 2D Overhead View?
    - If game is 3D as you have stated; what is the view style: First Person, 3rd Person, View and Zoom, Overhead, Moveable Camera with auto follow or fixed?
    - Is it an adventure game, a role playing game, first person shooter, strategy?
    - What are the rules of the game?
    - What are the challenges, achievements, goals, and scoring or rating systems?
    - Is it single player, multi player, or mass multi online player?
    - Does it have levels or boards, or is it a sandbox open world?
    - Is there a story, plot and main characters?
    - What is the ultimate achievement or the final goal?
    - Does the game infinitely progress?
    - What type of background music and sound effects should it have, such as fantasy fanfare, drum beats and trumpets such as a war march, mystical, suspense and horror?

- *Who and what is our target audience?*
    - Is this game for PC, Console, Tablet, Smartphone, Web browser?
    - Is it for Windows, Apple-Mac, Linux-Unix, Android Devices or all of them or just some but not others?
    - This deals with choosing the API to use either it be DirectX, OpenGL, or others -
        - DirectX - Which version? 9.0c, 10, 11, 12? This will affect target audience since different version work with different operating systems of windows such as DirectX 12 only works with Windows 10. Also not everyone has a DirectX 12 capable video card.
        - OpenGL - The version only makes a difference to some extent since almost all hardware vendors must support at least the minimal Core API library, yet they are each allowed to support their own independent list of supported extensions.
        - Others - Such as the newly formed Vulkan, the same concepts from above can be applied.
    - Is the target audience for younger children say between 3-7? Is it for children say 5-12? For teens 12 - 17? Mature for adults 17 or 18+?

        - Is the game geared to gender specific? Such as is it a game intended for boys or men such as wrestling, boxing etc., or is it more intended for girls and women such as Barbie Dress Up or Horse Grooming? This isn't meant to be stereotypical, just something to consider when designing a project.
    - *Choosing an API*
        - If you know your target audience is primarily using Windows, then 9/10 times DirectX is the choice to go, albeit Modern OpenGL works just as good!
        - If you know that you are targeting window users, but also want to include non-windows systems, then you may want to use OpenGL since DirectX isn't available for other operating systems. The best bet would be to use both API's and have a rendering switch based on the OS in use, or give the user the option during Game Installation if PC to choose which version of the Game such as DirectX or OpenGL. If it is Windows Phone or iPhone or Android, then OpenCL is probably your bet then.
    - *Choosing a programming language*
        - This all depends on the particular needs.
        - The most commonly used language for large scale 3D Game Engine projects is primarily found using C++ because of the language's

versatility in allowing you to do many different tasks not available in other languages such as managing your own memory dynamically, writing generic programming through templates, object oriented by using classes, inheritance and polymorphism or using a base classes to design abstract class at the root of a class hierarchy.

- C# is not without its limits for it does allow for easier code writing without having to worry about manually managing memory and other things.
- Java is still widely used as it's virtual machine is found on almost every device.
- JavaScript is excellent when working with any major browser such as IE11, Chrome, Firefox etc.
- Visual Basic is one of the least used, but it still has it's merits:
    - For example: There does exists a program to make 2D RPGs out there and it is called RPG Maker. With the use of text boxes, lists, radio buttons etc. VB would be a great fit to design the application part that makes the games, however it also has it's own scripting language that you can program game content from its own scripting language. In many cases these game scripting languages can be written from

almost any language such as C, C++, C#, Ruby, Pascal, Pearl, etc.

- Choosing appropriate 3rd Party Libraries that are well known, stable, reliable, trustworthy, constantly updated, flexible and portable. One such as GLM which is an open source math library that was written with the feel and style of OpenGL's GLSL shading language. It is constantly updated, works very well, is fully portable, easy installation as it is a headers only library; no need for other dependencies, binaries or having to recompile it for each different version of the compiler you are using etc.

- *The Assets*
  - What kind of theme does or will the game have?
    - The Art - Graphics, Textures, Models, etc. - The overall look and feel.
    - The Audio - Music background, sound effects etc. - The Music Compositions.
    - Storyline: - Cut Scenes, Video, Animation, Character Dialogs etc.
- *Game Control and Logic*
  - What kind of controls? - Keyboard & Mouse, Joystick, Hand Held Controller?
  - The actual motion of the Characters or Game Objects. Kinematic motion, point and click, point and drag, hover over, etc.

    - The Actions of the Characters or Game Pieces.
    - The logic of the game: If this then that etc. Scoring System or Goal Achievement
- *Math-Math & More Math* - Without it; Good Luck!
    - Basic Algebra
    - Geometry
    - Trigonometry
    - Logarithms, Exponentials, Summations and Matrices.
    - Points, Vectors and Matrices
    - Calculus - Differentiation: Derivatives, Partial Derivatives; Integration: Integrals, Integrands, both definite and indefinite integrals.
    - Linear Algebra
    - Vector Calculus
    - Vector Fields
    - Abstract Algebra
    - Analytical Geometry (Geometrical Projections) - Non Euclidean.
    - Physics - Kinematics, Quadratics, Forces, Vector Fields, Collisions, Explosions. Just for game content. Electrical Circuitry for circuit boards: need to know your systems and how ROM, RAM, CPU, CACHE, VRAM, and GPUS

all communicate across the BUS. Without this you can not "Optimize" your Game Engine!

The Tools Needed

- A high powered pc workstation that is capable of doing graphic, video and audio manipulation with ease.
- The appropriate OS with extended cards: Video, Audio, Networking etc.
- The programming languages and their compilers. C++, C#, Java etc. Any well known IDE.
- A graphics editor such as Photoshop, Gimp, Texture Editor etc.
- Modeling programs such as Maya, 3DS Max etc.
- Audio as there are two varieties:
    - Orchestra composition for making music scores to export into 3D Audio files.
    - Sound Effects to output audio types such as *.wav files.
- All of the appropriate 3rd party libraries to simplify the amount of code you will have to write.
- A Hex Editor for reading and manipulating Binary Files!
- A Programming Calculator - For doing both binary and hexadecimal calculations, words, and bytes etc.

- A versatile 3D graphing calculator for doing advanced mathematics. You may need to solve a system of equations with multiple variables to find either their derivatives, 2nd order derivatives, partial derivatives, integrals, doing trigonometric equations, vectors, and matrices. Especially Vectors and Matrices using trigonometry for doing rotations, and more advanced concepts such as quaternions and maybe even complex numbers although I have rarely seen the need for complex numbers in any Game Engine, except maybe for some concepts such as in particle physics and AI-programming.

The Concepts of a Game Engine

- *The Parts* - In todays industry a Game Engine isn't just one thing, it can be considered multiple little engines integrated together into one working engine.
    - *Graphics Engine* - Window Creation, and API setup for rendering Primitives to the screen each game frame while considering the monitors' refresh rate and its pixel resolution as well as the GPUs draw rate.
        - *Textures* - Normally a 2D graphic images stored in memory with a width and height either measured in inches or pixels, and can have various properties or attributes such as color encoding, and compression levels.

- *Sprites & Fonts* - Are usually associated with a texture since they are in a sense a specific type of texture.
- *Models* - Models are usually 3D Geometric Meshes composed of mostly triangles, sometimes lines, points and other primitives where a specified texture can be applied to them.
- *Particles* - Particles and even some aspects of animation can be applied here although the particle engines along with animations are more associated with the physics engine because they apply physics from the physics engine.
- *3D Environment* - Could be in a closed room, outdoors, or outer space or regular flight.
- *Camera* - The view angle at which the User will see the Game World From.
    - *First Person* - The illusion as if you were looking through the main characters eyes. The main character and camera turn and move together in unison.
    - *Third Person* - The main character is fully in front of the camera. The camera will rotate around the point of origin with that origin being the main

character unless independent controls are given, the camera will follow or translate linearly as the character moves. It may also have a zoom option. Also the tilted view is how much angel does the camera project between the main characters direct line of sight (horizontal) versus how much the camera is pointed towards the ground.

- *Materials* - The color attributes of a model's or a triangle's face: It is usually defined by 4 color channels (RGBA) where RGB are the Red Green and Blue channels and A is the alpha or the amount of transparency it has. Materials are important because they affect the overall shade of the object. If you have a model of a box loaded into memory that has an all yellow face and you apply a blue material to it, through color addition and subtraction it would now appear green. Materials are also useful and important when it comes to lighting which you will see in the next section which are described by these kind of material - lighting combinations: Ambient, Diffuse and Specular colors or material properties.
    - *Ambient* - The color and intensity that the light puts off.
    - *Diffuse* - The color saturation of the object itself (Self emitting)

        - *Specular* - The shininess of a region of the object that changes color from the light intensity on shiny type objects such as an apple or metal.
    - *Lighting* - Much can be said about this but without it you would not be able to see an interesting setting and you would have minimal visual effects.
        - *Point Light* - Has location but no direction as it shines out from a single point such as a light bulb.
        - *Directional Light* - Has direction but no location such as the sun
        - *Spot Light* - Has both direction and position as well as a cutoff angle. Such a light is like a flash light. It also attenuates.
        - *Properties* - Some lights attenuate, others have intensities, specular highlights, cutoff angles, etc.
    - *Shadow, Reflections etc.* - This is the ability to use the above lights materials, their colors, directions, and intensities etc. to cast either a shadow of an object or even to reflect objects such as the sky or clouds onto a surface of shiny metal or a body of water.

- *Weather Patterns* - Fog, Rain, Snow, Fires etc. Now Fog is usually done with shaders, and so are the others, but in most cases things such as rain, snow and fire are usually generated with a particle engine.
- *Motion - Transformations* - Some of these are tied into the game logic of the controls to be used, some are tided into the scene graph such that if 1 object has multiple parts and the whole object is translated all the parts are translated together, and some of these are tided in with the physics engine for things such as collision detections and even the AI system for enemies to check for boundaries and to be aware of the player as being hostile. While many of them maybe tied to multiple engine parts.
    - *Linear Transformations* - Transformations That Are Usually Considered 1D and sometimes 2D such as Rotations about a single point or single line or axis.
        - *Translations* - Walking, running etc.
        - *Scaling* - Stretching or Skewing objects
        - *Rotation* - Rotating with an angle about a single axis, vertex(point) or vector(line).

    - *Quadratic Transformations* - Throwing a Ball, Jumping etc.
    - *Affine Transformations* - Matrix Transformations going from one View to another such as going from model to view to projection and there are others such as orthographic view; especially going from a 3D structure and to transform the data to fit onto a flat 2D plane such as your screen without the presence of stretching, tearing, and artifacts.
    - *Complex Transformations* - Rotations done in 3D among multiple axis of rotations at a time. Using Euler Angles this has a side effect called Gimbal Lock; to avoid it, we use Quaternions. Which is 4D mathematics involving matrices and complex numbers.
- *Scene Graph* - Bringing them together to produce a working scene of related objects that defines the current world with all of its parts and where one object is in position related to another. There are many types of these graphs: BSP, Root - Node with leafs, Marching Cubes, Cubic or Octal Space Partitioning systems, advanced hashing table techniques etc.
- *The HUD* - Heads Up Display

    - *GUI-System* - Graphical User Interface (Buttons, List Boxes, Text Boxes etc.)
    - *Text Rendering System* - How different fonts will be rendered.
- *Physics Engine* - The Engine that keeps the game's refresh rate of its drawing calls to be in sync with both the monitors refresh rate and the GPU's drawing rate. It also does all the basic calculations for the majority of general motion that can be applied to any non static game object. It is also used in doing collision detections, ray tracing, and used for doing animations.
- *Animation Engine* - Usually a collection of character and object animations that are predefined to some default value yet may be modified by a scripting language if such a scripting process is designed to be a part of the overall engine. (Better know how to write a parser and a compiler if you decide to include your own scripting language)
- *Audio Engine* - Usually the controller that knows when to play the expected audio file either it be a background music, transitional music, or some sound effect. Also the ability to apply 3D Audio. Audio with multiple channels to apply specific sounds to a specific direction or location. Also the ability to not just play audio from a file, but from a Stream Object.

- *AI Engine* - The engine that controls the abilities of all NPCs (Non Playable Characters). Their paths, their goals, their hostility levels, choices of attacks, the dialogue of conversations with the Player etc.
- *Networking Engine* - The engine responsibility to creating network type data packets of needed game information and to transmit them over the network between a server and a client. The ability to both write and read(parse) the packets.
- *Shader Engine* - Engine responsible for handling all of the Shaders
- *Memory Management System* - Engine responsible for handling all of the games assets in memory.

Putting it All Together

One of the most sought out design of a versatile game engine in the modern industry is to have a working engine or set of engines that works independently from any Game Content, Game Properties, and Game Logic so that a Game Developer can rip out the old game with all of its source code and game assets and the Engine doesn't break to make another game using the same existing engine. Making it modular and portable and as generic as possible is another important aspect. Albeit there may be some modifications to the engine from one Game Style to another. Such as the difference between a

First Person Shooter and an MMORPG. Special Care needs to be taken when beginning to work in this field for there are a lot of things to consider such as the topic of Batch Rendering.

Once you have a working 3D-2D Game Engine with all of its various parts that is nearly bug free and fully optimized then the next step is to get involved with your other applications for content creation where you will need to make your textures (or download free ware ones), make your models, create your levels or immediate map view of the world, create your sound effects and music or (download free ware). Then plan out the game logic with a starting point, ending point and goals for different things in between. Of course you could also pay to buy some of that content or pay others to do some of the work for you. This is why it takes studios such as EA, Valve, Blizzard, Bethesda etc. a few years with thousands of people to make any decent 3D Game either it be for the Play Station, X-Box, or PC systems. But this is where to start! The Things to Consider and What it actually Takes to Make a Working 3D Game!

This is only the tip of the Iceberg too for there are many other concepts that I left out or didn't cover such as the different types of algorithms for doing specific tasks such as the order of rending objects base on distance from camera due to transparencies. I didn't even cover things such as error handlers, exception handlers, logging information to a file or screen during debugging and building modes before final release version, multi-

threading, parallel programming, even working down to ASM instruction sets, concurrency, etc.

The bottom line is the most important thing you need into making a game is the dedication to learn and then to try and fail, fail, fail and fail again, but to get back up and try again until you accomplish it. For this you will need 1,000s and 1,000s of Hours! Researching, Programming, DEBUGGING, DEBUGGING, DEBUGGING! Then Repeat it all again! Why Repeat? Well once you get the hang of it; that technology is old, and you have to learn about the new technology and the new compilers, and the new APIs etc.!

If you want to know the inner workings of a basic complete 3D Game Engine from scratch check out this website MarekKnows and you can follow his series of video tutorials. His projects are primarily on windows using MSVS's IDE's and Compilers and he focuses on OpenGL. Once you have a grasp of the concepts of the Game Engine and later in a newer series Shader Engine, you can easily take those concepts and apply them to DirectX if you so choose, but you will have to change the Windows Code to support DirectX as opposed to OpenGL, and then learn how to use their API in place of OpenGL. The two are quite different!

edited Feb 10, 2017 at 8:46

answered Feb 3, 2017 at 6:52

Francis Cugler
7,865 ● 2 ● 35 ● 66

As a tip to future readers... MarekKnows website is no longer active. You can dow a MarekKnows or Marek Krzemenski(or something like that) search on Youtube... He only has a few of his introductory or free videos... A lot of his content was not free! If you want to learn how to do basic 3D Scene Rendering and not a full scale Game Engine and then Game... You can check www.learnopengl.com that's a really good site. – Francis Cugler Mar 11, 2021 at 1:30

0

StackOverflow etiquette means you'll get closed as "too broad" for a question. Specific questions are best here.

If you are mostly interested in "just writing a game", then you should consider using an existing engine like Unity 3D or Unreal Engine. Unity 3D is C#, which is a very common language in business development and is reasonably popular for mobile development. Unreal Engine is UnrealScript and C++ so is more common for console and PC development. Both have large, vibrant online communities. There are of course lots of other engines and game toolkits out there, so don't consider this an exhaustive listing.

If you are more interested in expanding your knowledge of computer programming through the effort of writing a game 'from scratch', or if you are hoping to become an expert in graphics and/or game technologies, then C++ is a better route. To get started with DirectX 11 in C++, you

should look at DirectX Tool Kit which provides a good starting place for hacking together your own DirectX game engine.

> If you find the C++ language barrier a bit too much, but want something more 'techy', then take a look at SharpDX .NET managed assemblies for using DirectX APIs. See DirectX and .NET.

For some book recommendations for learning DirectX 11, see this blog post. Be aware that many of the online tutorials (including the very popular Rastertek tutorials) are somewhat dated as they make heavy use of the legacy DirectX SDK. For details on that exactly that means when using Visual Studio 2015, see MSDN and Where is the DirectX SDK (2015 Edition)?.

C++ is a bit like the game of Go: It's relative easy to pick up the basics, but it takes years of experience to really master it. C++ provides an immense level of control, and most game engines are written in C++ at some level. You can integrate easily with the large collection existing C/C++-based libraries, and it's supported for almost all gaming platforms. All that said, the language is quite complex, and can be challenging to learn the best coding patterns as well as develop the robust debugging skills and discipline needed to be highly productive with it.

> BTW, there's a project focused on making C++ easier to learn you might want to check out, the C++ Core Guidelines.
>
> *Within C++ is a smaller, simpler, safer language struggling to get out.* -- Bjarne Stroustrup

Share Improve this answer Follow

edited Jun 20, 2020 at 9:12

Community Bot

1 ● 1

answered Feb 3, 2017 at 3:54

Chuck Walbourn

40.9k ● 2 ● 67 ● 90