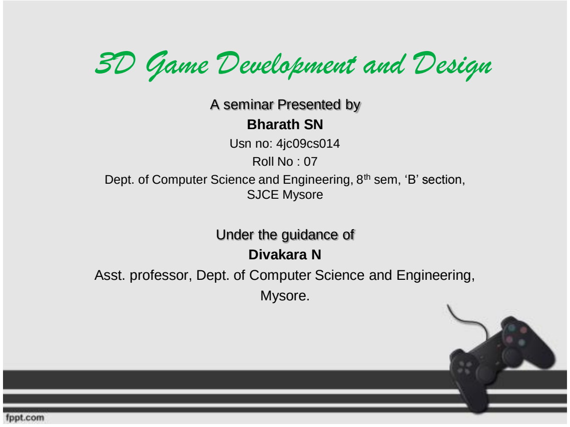
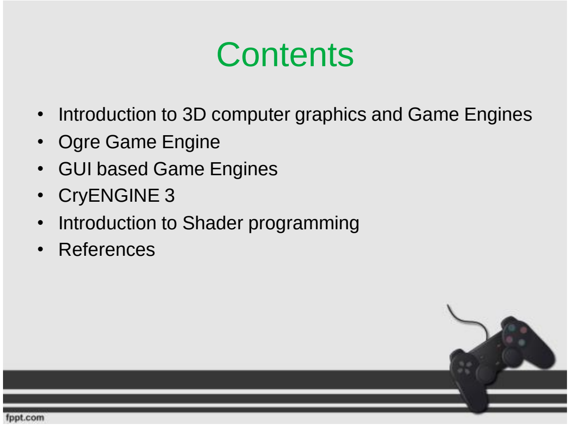


3D Game Development and Design

A seminar Presented by
Bharath SN
Usn no: 4jc09cs014
Roll No : 07
Dept. of Computer Science and Engineering, 8th sem, 'B' section,
SJCE Mysore

Under the guidance of
Divakara N
Asst. professor, Dept. of Computer Science and Engineering,
Mysore.

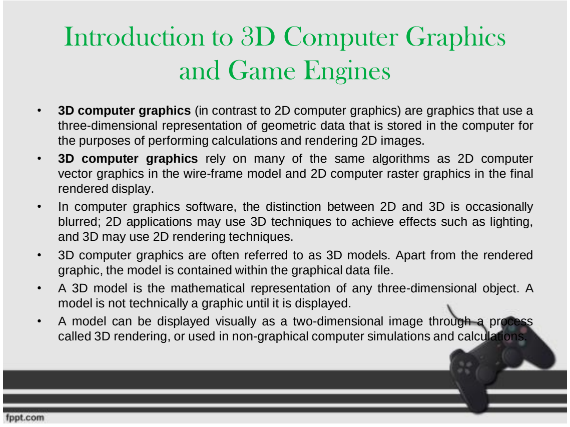
fppt.com



Contents

- Introduction to 3D computer graphics and Game Engines
- Ogre Game Engine
- GUI based Game Engines
- CryENGINE 3
- Introduction to Shader programming
- References

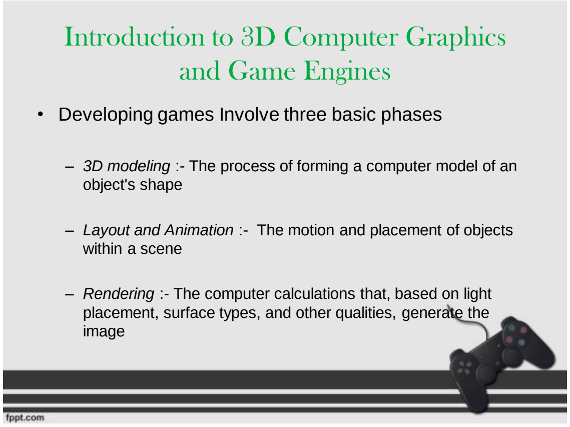
fppt.com



Introduction to 3D Computer Graphics and Game Engines

- **3D computer graphics** (in contrast to 2D computer graphics) are graphics that use a three-dimensional representation of geometric data that is stored in the computer for the purposes of performing calculations and rendering 2D images.
- **3D computer graphics** rely on many of the same algorithms as 2D computer vector graphics in the wire-frame model and 2D computer raster graphics in the final rendered display.
- In computer graphics software, the distinction between 2D and 3D is occasionally blurred; 2D applications may use 3D techniques to achieve effects such as lighting, and 3D may use 2D rendering techniques.
- 3D computer graphics are often referred to as 3D models. Apart from the rendered graphic, the model is contained within the graphical data file.
- A 3D model is the mathematical representation of any three-dimensional object. A model is not technically a graphic until it is displayed.
- A model can be displayed visually as a two-dimensional image through a process called 3D rendering, or used in non-graphical computer simulations and calculations.

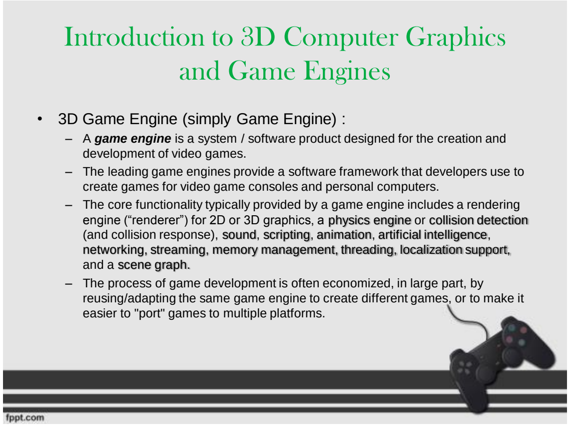
fppt.com



Introduction to 3D Computer Graphics and Game Engines

- Developing games Involve three basic phases
  - *3D modeling* :- The process of forming a computer model of an object's shape
  - *Layout and Animation* :- The motion and placement of objects within a scene
  - *Rendering* :- The computer calculations that, based on light placement, surface types, and other qualities, generate the image

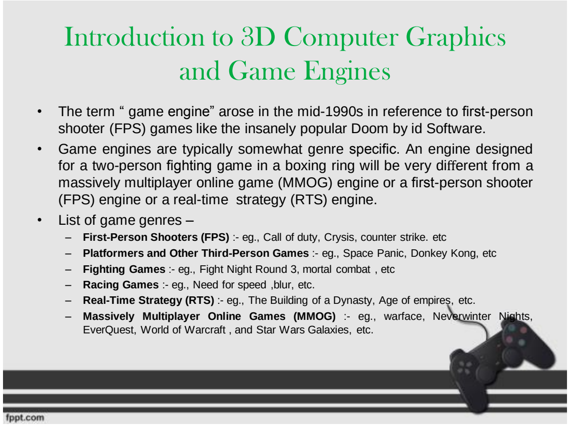
fppt.com



Introduction to 3D Computer Graphics and Game Engines

- 3D Game Engine (simply Game Engine) :
  - A ***game engine*** is a system / software product designed for the creation and development of video games.
  - The leading game engines provide a software framework that developers use to create games for video game consoles and personal computers.
  - The core functionality typically provided by a game engine includes a rendering engine ("renderer") for 2D or 3D graphics, a physics engine or collision detection (and collision response), sound, scripting, animation, artificial intelligence, networking, streaming, memory management, threading, localization support, and a scene graph.
  - The process of game development is often economized, in large part, by reusing/adapting the same game engine to create different games, or to make it easier to "port" games to multiple platforms.

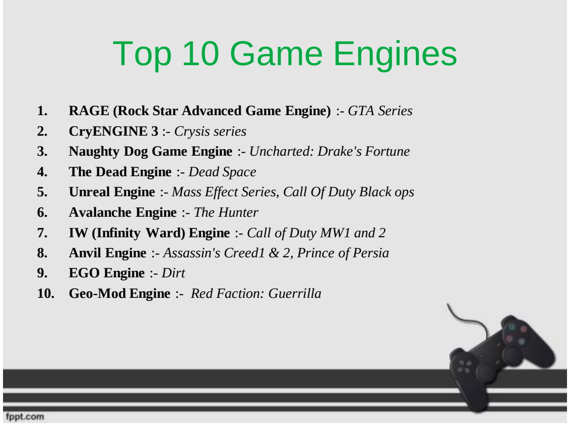
fppt.com



Introduction to 3D Computer Graphics and Game Engines

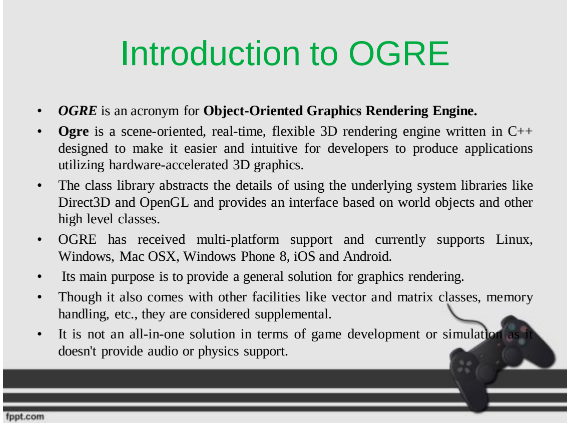
- The term " game engine" arose in the mid-1990s in reference to first-person shooter (FPS) games like the insanely popular Doom by id Software.
- Game engines are typically somewhat genre specific. An engine designed for a two-person fighting game in a boxing ring will be very different from a massively multiplayer online game (MMOG) engine or a first-person shooter (FPS) engine or a real-time strategy (RTS) engine.
- List of game genres –
  - **First-Person Shooters (FPS)** :- eg., Call of duty, Crysis, counter strike. etc
  - **Platformers and Other Third-Person Games** :- eg., Space Panic, Donkey Kong, etc
  - **Fighting Games** :- eg., Fight Night Round 3, mortal combat , etc
  - **Racing Games** :- eg., Need for speed ,blur, etc.
  - **Real-Time Strategy (RTS)** :- eg., The Building of a Dynasty, Age of empires, etc.
  - **Massively Multiplayer Online Games (MMOG)** :- eg., warface, Neverwinter Nights, EverQuest, World of Warcraft , and Star Wars Galaxies, etc.

fppt.com



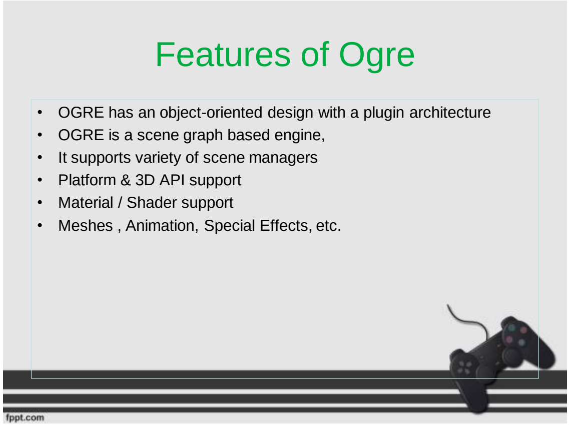
Top 10 Game Engines

1. **RAGE (Rock Star Advanced Game Engine)** :- *GTA Series*
2. **CryENGINE 3** :- *Crysis series*
3. **Naughty Dog Game Engine** :- *Uncharted: Drake's Fortune*
4. **The Dead Engine** :- *Dead Space*
5. **Unreal Engine** :- *Mass Effect Series, Call Of Duty Black ops*
6. **Avalanche Engine** :- *The Hunter*
7. **IW (Infinity Ward) Engine** :- *Call of Duty MW1 and 2*
8. **Anvil Engine** :- *Assassin's Creed1 & 2, Prince of Persia*
9. **EGO Engine** :- *Dirt*
10. **Geo-Mod Engine** :- *Red Faction: Guerrilla*



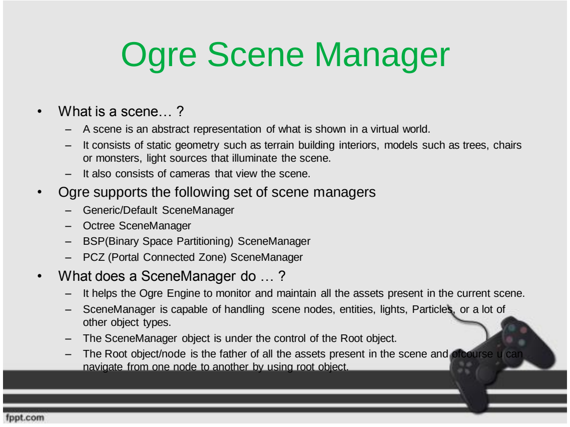
Introduction to OGRE

- ***OGRE*** is an acronym for **Object-Oriented Graphics Rendering Engine.**
- **Ogre** is a scene-oriented, real-time, flexible 3D rendering engine written in C++ designed to make it easier and intuitive for developers to produce applications utilizing hardware-accelerated 3D graphics.
- The class library abstracts the details of using the underlying system libraries like Direct3D and OpenGL and provides an interface based on world objects and other high level classes.
- OGRE has received multi-platform support and currently supports Linux, Windows, Mac OSX, Windows Phone 8, iOS and Android.
- Its main purpose is to provide a general solution for graphics rendering.
- Though it also comes with other facilities like vector and matrix classes, memory handling, etc., they are considered supplemental.
- It is not an all-in-one solution in terms of game development or simulation as it doesn't provide audio or physics support.



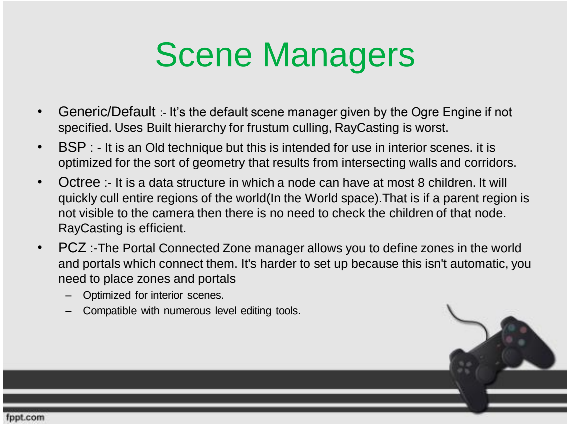
Features of Ogre

- OGRE has an object-oriented design with a plugin architecture
- OGRE is a scene graph based engine,
- It supports variety of scene managers
- Platform & 3D API support
- Material / Shader support
- Meshes , Animation, Special Effects, etc.



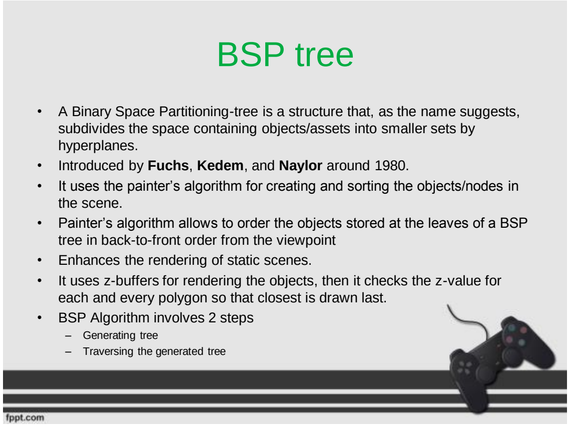
Ogre Scene Manager

- What is a scene… ?
  - A scene is an abstract representation of what is shown in a virtual world.
  - It consists of static geometry such as terrain building interiors, models such as trees, chairs or monsters, light sources that illuminate the scene.
  - It also consists of cameras that view the scene.
- Ogre supports the following set of scene managers
  - Generic/Default SceneManager
  - Octree SceneManager
  - BSP(Binary Space Partitioning) SceneManager
  - PCZ (Portal Connected Zone) SceneManager
- What does a SceneManager do … ?
  - It helps the Ogre Engine to monitor and maintain all the assets present in the current scene.
  - SceneManager is capable of handling scene nodes, entities, lights, Particles, or a lot of other object types.
  - The SceneManager object is under the control of the Root object.
  - The Root object/node is the father of all the assets present in the scene and ofcourse u can navigate from one node to another by using root object.



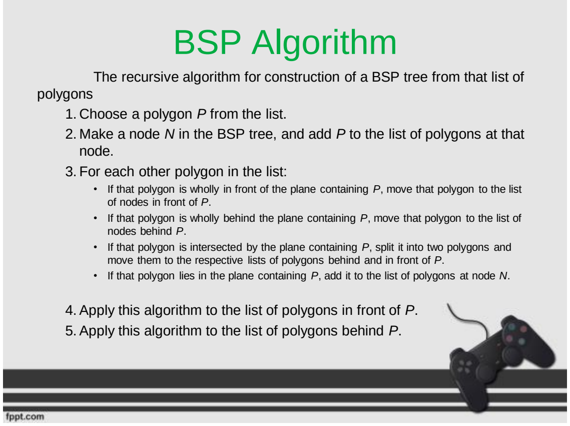
Scene Managers

- Generic/Default :- It's the default scene manager given by the Ogre Engine if not specified. Uses Built hierarchy for frustum culling, RayCasting is worst.
- BSP : - It is an Old technique but this is intended for use in interior scenes. it is optimized for the sort of geometry that results from intersecting walls and corridors.
- Octree :- It is a data structure in which a node can have at most 8 children. It will quickly cull entire regions of the world(In the World space).That is if a parent region is not visible to the camera then there is no need to check the children of that node. RayCasting is efficient.
- PCZ :-The Portal Connected Zone manager allows you to define zones in the world and portals which connect them. It's harder to set up because this isn't automatic, you need to place zones and portals
  - Optimized for interior scenes.
  - Compatible with numerous level editing tools.



BSP tree

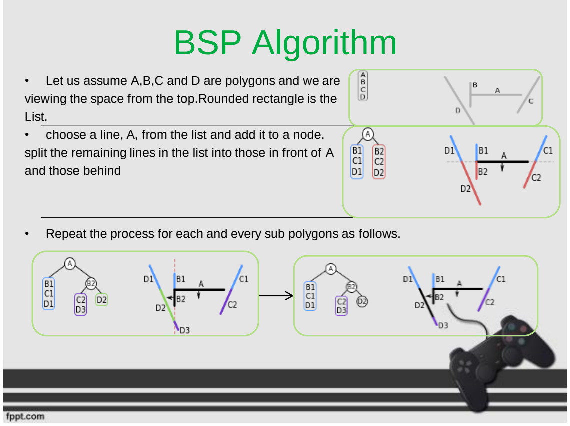
- A Binary Space Partitioning-tree is a structure that, as the name suggests, subdivides the space containing objects/assets into smaller sets by hyperplanes.
- Introduced by **Fuchs**, **Kedem**, and **Naylor** around 1980.
- It uses the painter's algorithm for creating and sorting the objects/nodes in the scene.
- Painter's algorithm allows to order the objects stored at the leaves of a BSP tree in back-to-front order from the viewpoint
- Enhances the rendering of static scenes.
- It uses z-buffers for rendering the objects, then it checks the z-value for each and every polygon so that closest is drawn last.
- BSP Algorithm involves 2 steps
  - Generating tree
  - Traversing the generated tree



BSP Algorithm

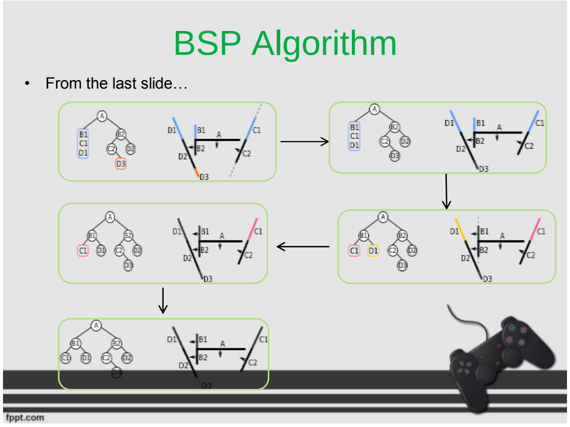
The recursive algorithm for construction of a BSP tree from that list of polygons

1. Choose a polygon *P* from the list.
2. Make a node *N* in the BSP tree, and add *P* to the list of polygons at that node.
3. For each other polygon in the list:
   - If that polygon is wholly in front of the plane containing *P*, move that polygon to the list of nodes in front of *P*.
   - If that polygon is wholly behind the plane containing *P*, move that polygon to the list of nodes behind *P*.
   - If that polygon is intersected by the plane containing *P*, split it into two polygons and move them to the respective lists of polygons behind and in front of *P*.
   - If that polygon lies in the plane containing *P*, add it to the list of polygons at node *N*.
4. Apply this algorithm to the list of polygons in front of *P*.
5. Apply this algorithm to the list of polygons behind *P*.



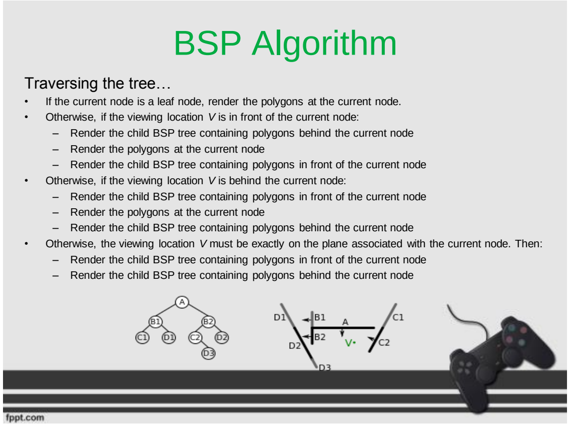
BSP Algorithm

- Let us assume A,B,C and D are polygons and we are viewing the space from the top.Rounded rectangle is the List.
- choose a line, A, from the list and add it to a node. split the remaining lines in the list into those in front of A and those behind
- Repeat the process for each and every sub polygons as follows.



BSP Algorithm

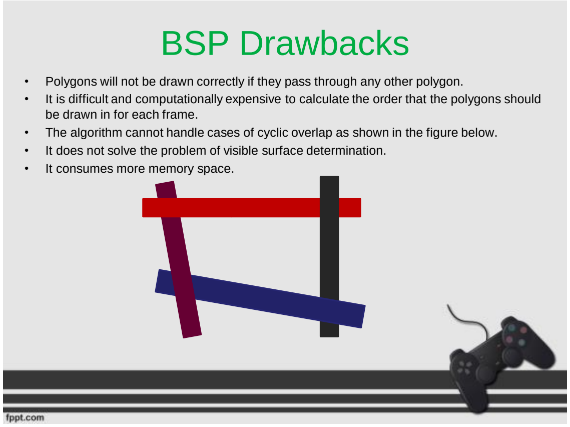
- From the last slide…



BSP Algorithm

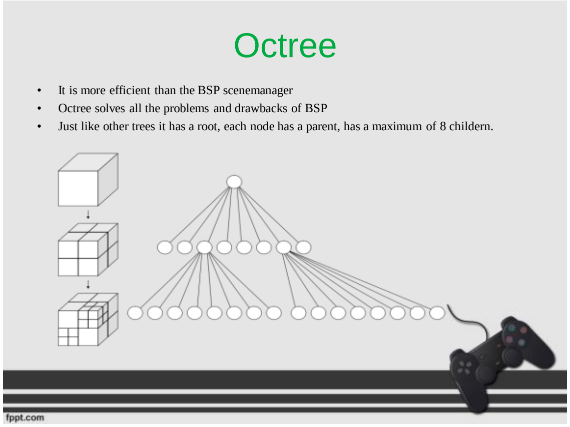
Traversing the tree…

- If the current node is a leaf node, render the polygons at the current node.
- Otherwise, if the viewing location *V* is in front of the current node:
  - Render the child BSP tree containing polygons behind the current node
  - Render the polygons at the current node
  - Render the child BSP tree containing polygons in front of the current node
- Otherwise, if the viewing location *V* is behind the current node:
  - Render the child BSP tree containing polygons in front of the current node
  - Render the polygons at the current node
  - Render the child BSP tree containing polygons behind the current node
- Otherwise, the viewing location *V* must be exactly on the plane associated with the current node. Then:
  - Render the child BSP tree containing polygons in front of the current node
  - Render the child BSP tree containing polygons behind the current node



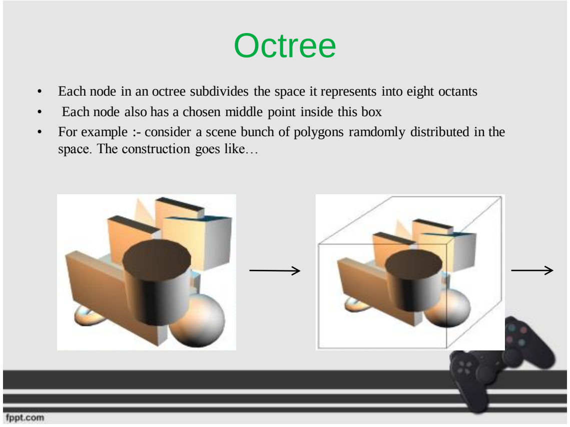
BSP Drawbacks

- Polygons will not be drawn correctly if they pass through any other polygon.
- It is difficult and computationally expensive to calculate the order that the polygons should be drawn in for each frame.
- The algorithm cannot handle cases of cyclic overlap as shown in the figure below.
- It does not solve the problem of visible surface determination.
- It consumes more memory space.



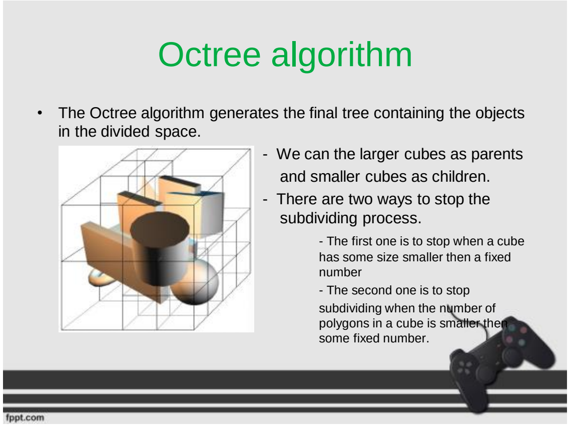
Octree

- It is more efficient than the BSP scenemanager
- Octree solves all the problems and drawbacks of BSP
- Just like other trees it has a root, each node has a parent, has a maximum of 8 childern.



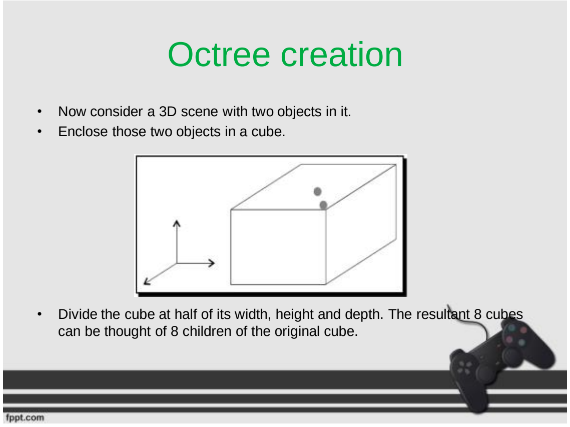
Octree

- Each node in an octree subdivides the space it represents into eight octants
- Each node also has a chosen middle point inside this box
- For example :- consider a scene bunch of polygons ramdomly distributed in the space. The construction goes like…



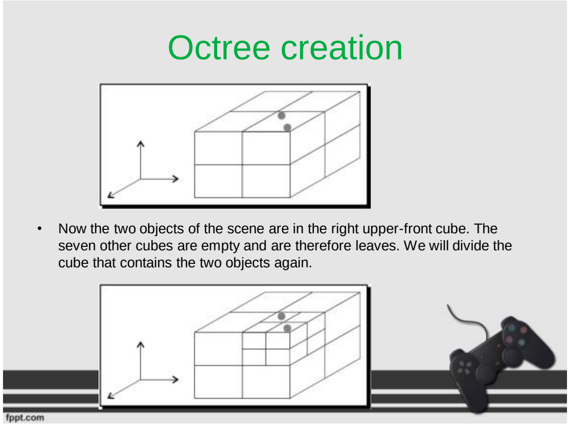
Octree algorithm

- The Octree algorithm generates the final tree containing the objects in the divided space.
  - We can the larger cubes as parents and smaller cubes as children.
  - There are two ways to stop the subdividing process.
    - The first one is to stop when a cube has some size smaller then a fixed number
    - The second one is to stop subdividing when the number of polygons in a cube is smaller then some fixed number.



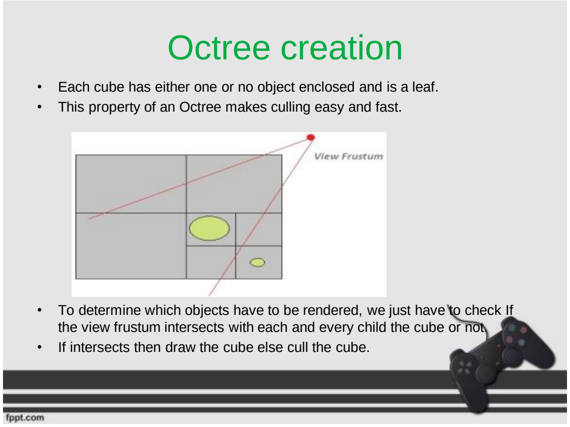
Octree creation

- Now consider a 3D scene with two objects in it.
- Enclose those two objects in a cube.
- Divide the cube at half of its width, height and depth. The resultant 8 cubes can be thought of 8 children of the original cube.



Octree creation

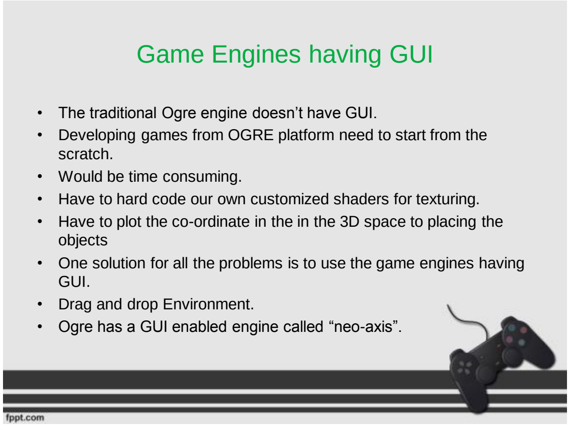
- Now the two objects of the scene are in the right upper-front cube. The seven other cubes are empty and are therefore leaves. We will divide the cube that contains the two objects again.



Octree creation

- Each cube has either one or no object enclosed and is a leaf.
- This property of an Octree makes culling easy and fast.

- To determine which objects have to be rendered, we just have to check If the view frustum intersects with each and every child the cube or not
- If intersects then draw the cube else cull the cube.



Game Engines having GUI

- The traditional Ogre engine doesn’t have GUI.
- Developing games from OGRE platform need to start from the scratch.
- Would be time consuming.
- Have to hard code our own customized shaders for texturing.
- Have to plot the co-ordinate in the in the 3D space to placing the objects
- One solution for all the problems is to use the game engines having GUI.
- Drag and drop Environment.
- Ogre has a GUI enabled engine called “neo-axis”.

CryENGINE 3

- The CryENGINE 3 is a game engine which drives the visual actions taking place on the screen.
- It is a wholly “What you See Is What U Play(WYSIWIP)” technology.
- Game Engine designed by Crytek.
- The most prominent tool provided by a game engine is the level. This is done by CryENGINE 3 sandbox.
- It contains Each and every tools needed for level designing like, assets, models, animations, texturing, Video rendering, Audio subsystem, AI subsystem, material editor, etc…
- It is specifically designed for first-person shooter genre.
- Freely available at www.crydev.net .

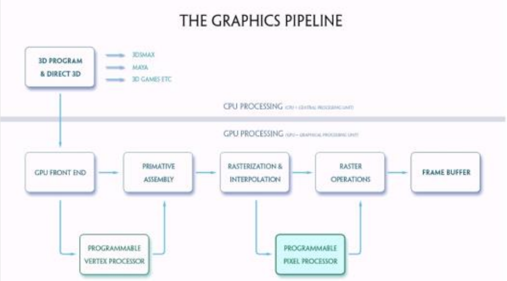
CryENGINE 3

(our college lobby)

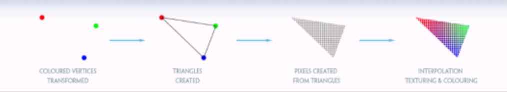
Introduction to Shader Programming

- A shader is a computer program that runs on the graphics processing unit and is used to do shading.
- A programmer can think of the main program as being executed just once on a CPU. But a shader program is executed repeatedly on a GPU.
- Executed for each and every elements of data in a stream.
- Shaders calculate rendering effects on graphics hardware with a high degree of flexibility
- The material editor auto-generates the shader code during texturing.
- There are three types of shading languages available
  - HLSL(High level shading language) / Direct 3D
  - GLSL (Open GL shading language)
  - CG (by NVIDIA and Microsoft)
- Types of Shaders
  - vertex Shaders
  - Geometry shaders
  - Fragments/pixel shaders.

Shaders(GPU pipeline)



Shaders (Data flow)

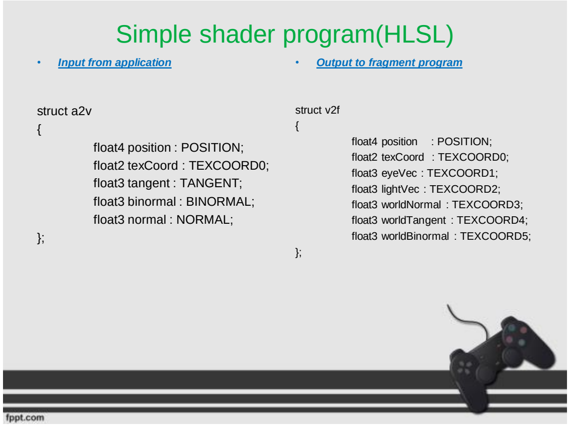

COLOURED VERTICES TRANSFORMED
TRIANGLES CREATED
PIXELS CREATED FROM TRIANGLES
INTERPOLATION TEXTURING & COLOURING

Simple shader program(HLSL)

- ***UI elements***

```
float4 AmbientColor : Ambient
<
   string UIName = "Ambient Color";
➢   = {0.25f, 0.25f, 0.25f, 1.0f};

float4 DiffuseColor : Diffuse
<
   string UIName = "Diffuse Color";
> = {1.0f, 1.0f, 1.0f, 1.0f};


float4 SpecularColor : Specular
<
   string UIName = "Specular Color";
> = { 0.2f, 0.2f, 0.2f, 1.0f };
```

```
float Glossiness <
           string UIWidget = "slider";
   int UIMin = 1;
   int UIMax = 128;
   int UIStep = 1;
   int UIStepPower = 16;
   string UIName = "Glossiness";
➢    = 40;

texture diffuseMap : DiffuseMap
<
   string name = "default_color.dds";
           string UIName = "Diffuse Texture";
   string TextureType = "2D";
>;

texture normalMap : NormalMap
<
   string name = "default_bump_normal.dds";
           string UIName = "Normal Map";
   string TextureType = "2D";
>;
```



Simple shader program(HLSL)

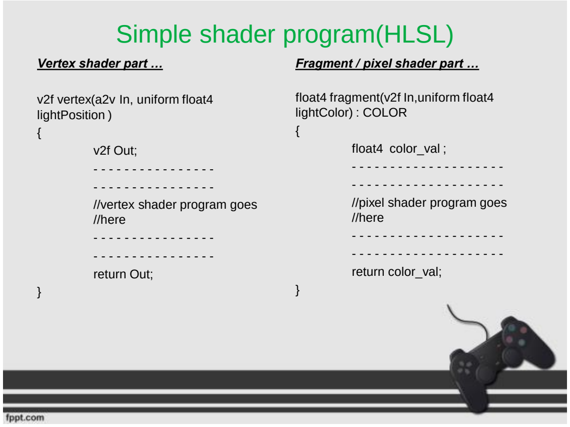
- ***Input from application***

```
struct a2v
{
        float4 position : POSITION;
        float2 texCoord : TEXCOORD0;
        float3 tangent : TANGENT;
        float3 binormal : BINORMAL;
        float3 normal : NORMAL;
};
```

- ***Output to fragment program***

```
struct v2f
{
        float4 position    : POSITION;
        float2 texCoord   : TEXCOORD0;
        float3 eyeVec : TEXCOORD1;
        float3 lightVec : TEXCOORD2;
        float3 worldNormal  : TEXCOORD3;
        float3 worldTangent  : TEXCOORD4;
        float3 worldBinormal  : TEXCOORD5;
};
```

fppt.com



Simple shader program(HLSL)

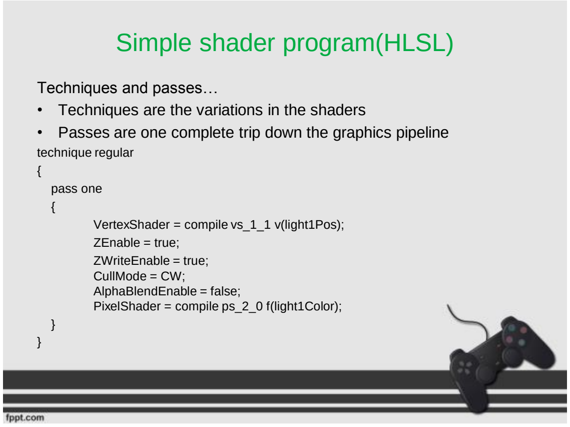
Vertex shader part …

```
v2f vertex(a2v In, uniform float4
lightPosition )
{
        v2f Out;
        - - - - - - - - - - - - - - - -
        - - - - - - - - - - - - - - - -
        //vertex shader program goes
        //here
        - - - - - - - - - - - - - - - -
        - - - - - - - - - - - - - - - -
        return Out;
}
```

Fragment / pixel shader part …

```
float4 fragment(v2f In,uniform float4
lightColor) : COLOR
{
        float4  color_val ;
        - - - - - - - - - - - - - - - - - - - -
        - - - - - - - - - - - - - - - - - - - -
        //pixel shader program goes
        //here
        - - - - - - - - - - - - - - - - - - - -
        - - - - - - - - - - - - - - - - - - - -
        return color_val;
}
```

fppt.com



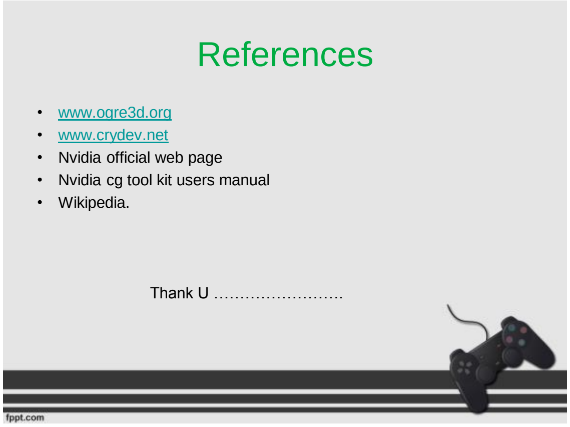
Simple shader program(HLSL)

Techniques and passes…

- Techniques are the variations in the shaders
- Passes are one complete trip down the graphics pipeline

```
technique regular
{
   pass one
   {
        VertexShader = compile vs_1_1 v(light1Pos);
        ZEnable = true;
        ZWriteEnable = true;
        CullMode = CW;
        AlphaBlendEnable = false;
        PixelShader = compile ps_2_0 f(light1Color);
   }
}
```

fppt.com



References

- www.ogre3d.org
- www.crydev.net
- Nvidia official web page
- Nvidia cg tool kit users manual
- Wikipedia.

Thank U ……………………..

fppt.com