IIWU : IF I WERE YOU
A SIMPLE GAMEPLAY UNMANAGEABLE BY GAME ENGINES

Yann Creach, Alexandre Topol
Centre d'Etudes et de Recherche en Informatique (CEDRIC)
Conservatoire National des Arts & Métiers (CNAM)
292 rue St-Martin, 75003 Paris, France
yann@iiwu.org, topol@cnam.fr

KEYWORDS

Video Games, Real-time 3D rendering, OpenGL.

ABSTRACT

The originality of the game that we present comes from its many technical constraints usually distributed on various types of games. It integrates well known specificities of several kinds without really being in any of these categories. Game engines are influenced by the types of games since they propose many technical optimizations to accelerate their operations. In return, only well categorized games can be developed using game engines. Our game does not claim many resources but it requires original operations that can't be achieved by game engines. We show that the design of an original creative game difficult when the needs are vast and the means offered are reduced. The problem posed by IIWU is a beginning of answer to the question "why is there only very few original games ?". The answer that we try to outline is that this problem will remain a technical problem as long as there is no software for designing and generating games detached from those we see usually. We illustrate this fact with a graphical implementation problem: the mixing of multiple spotlights.

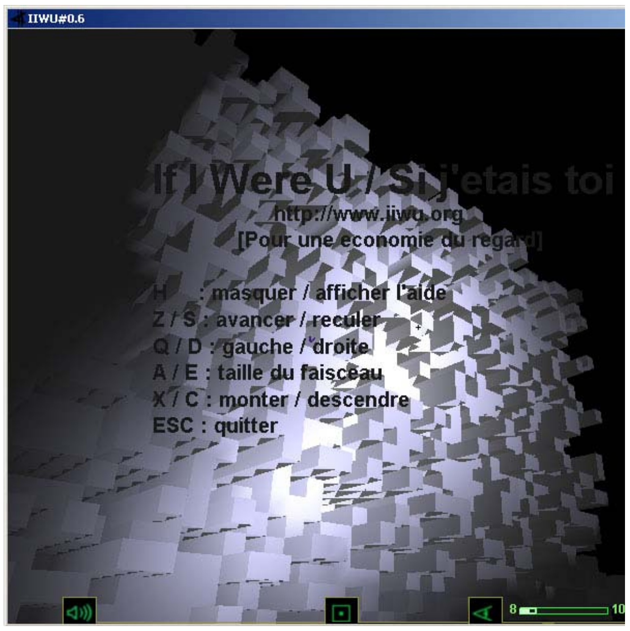
INTRODUCTION

The first 3D games are born in the early 90's. They can be seen like the creative appropriation of new technologies. Wolfenstein 3D is the game that created the First Person Shooter (FPS) kind. The 3D gaming technology was all new and graphic cards back then didn't help much to speed the rendering. The success of this game lies in the perfect osmosis between its scenario and the technology used to program it. Indeed, many simplifications were made to reduce the 3D rendering time and achieve a nice level of reactivity and fluidity needed for a real time playing: characters in 2D, all walls perpendicular to each others, restricted orientation of the camera.

Game engines evolved by gaining in immersive capacity, in sound synthesis and among all by exploiting the latest technological improvements of graphic materials. Thank to new techniques and to movie-like productions, new kinds of games have emerged through years. Racing games take advantage of a physical simulation engine, making it possible to show vehicles with realistic physical behaviors. RPGs connect many players on the Internet and do not suffer from its latency since they are played turn by turn. Hence, they don't need real time actions and thus adapt very well to the Internet. In a FPS, the 3D rendering is accelerated by different optimizations and the time saved can be used for cool visual effects.

Performances reached by these different types of games reside in the intensive optimization of the engine to match the gameplay. The game engine is specialized in order to achieve fast operations useful for the game (Eberly 2000). Progressively, the kind of game, the gameplay and the game engine became more and more closely connected.

The game that we present in this paper can't be categorized into one of the actual types of games. Hence, it can't take advantage of game engines optimized for those well known categories. Thus, our game can't benefit from their optimizations and paradigms. No existing game engine can help for the realization of such a game. The purpose of this article is to show the difficulty to create games which can not be classified in usual types. In this paper, we mainly discuss about the difficulty obtaining the wanted graphical effects.



If I Were U / Si j'etais toi
http://www.iiwu.org
[Pour une economie du regard]

H : masquer / afficher l'aide
Z / S : avancer / reculer
Q / D : gauche / droite
A / E : taille du faisceau
X / C : monter / descendre
ESC : quitter

Fig. 1: Screenshot of the game in its current version.

To meet game designers needs, some games require a heavy development almost from scratch. An important time is spent to study various techniques (not present in game engines) that can be used to achieve the wanted effects. The example we give with our game shows the difficulty for game designers to propose new creative and unseen types of games because they require a long development time. On one hand, game engines surely ease the development process but, on the other hand, they prevent developers to reach the game imagined by creative thoughts.

THE GAME

Presentation

IIWU - If I were you - is a play environment which depicts an "economy of the glance" to practice collectively on the Web. This environment plunged into the darkness materializes the glances of the multiple users by luminous projections. Each one can perceive spotlights driven by others and thus can identify other players and see what they are looking at. Above all, a user will see its beam of light as a sensory vector. But in the game, glances materialized in this way, are also transformed into "resources" that can allow trades. IIWU develops its playing dimension with this original management of glances, taken into account and convertible into “pov”.

The POV (Point Of View) is the currency and the trading unit within the game. It expresses both the concept of point of view in the majority of the games (the camera) and the concept of currency in the real life. Let us compare it boldly, but for a matter of clearness, with oil. Oil is a geographically distributed resource. It is also the raw material of many objects, an energy source convertible in driving energy for the majority of the terrestrial, air and marine vehicles, in calorific energy for the power plants and finally in electric power. It is a limited resource, object of geo-strategic fights and a stock exchange phenomenon.

The arena is made up of cubes laid out in space. All players can see the whole arena. They can move within the arena but they can not cross the cubes. The arena is plunged in a penumbra and players are equipped with a torch in order to perceive the scene and interact with it. The impact of the beam materializes the player moving on the surface of the cubes. The ray of light can be seen like the sword of the player, the weapon with which he/she will fight the other competitors.

The goal of the game is to gain as much POV as possible to become the master of the arena. To gain POV, the player can attack other players in a duel (a beam to beam fight) or appropriate cubes by converting them with his/her light beam.

Specificities of the game

As stated previously, the game is based on interactions between light beams (players avatars) and cubes positioned in a dark 3D space. This arena is shared between several players who fight through Internet. It is thus like a multi players FPS game. It requires a very short response time since all is based on remote displacements and interactions.

Usually, in a game, the world is static and can’t be modified. Hence, it is possible to achieve fast collision detections and culling techniques with algorithms like Binary Space Partitions (Naylor et al. 1990) or Octrees (Moore and Wilhelms 1988, Noborio et al. 1989) In our game, the arena is dynamic since cubes can be moved by players. Hence, the state of the arena must be stored into a database in order to know where and to who belongs a cube. It allows the game to run in a non-continuous way but in several sessions like with a RPG (Role Playing Game). Players can also move freely around the cubes like in a flight simulation game.

Multiplying the number of players means multiplying the number of lights. Managing an important number of dynamic lights hasn’t been done in recent games and can’t be done by game engines. Only static light maps and environment maps together with a few real dynamic lights can be integrated in a game. Hence, a new game engine must be written for our game.

Specific Game Engine

Technical aspects are quite as constraining. The game is Open Source and must be executed on the three main platforms suitable for internet connection, i.e. Linux, Mac and Windows. Creating a 3D game engine requires to take into account various aspects. The 3D engine, as we can see in figure 2, is an essential part but it is not sufficient to act like a game engine. A game integrates very different aspects like the game rules, the physical simulation, sounds, user actions, etc.

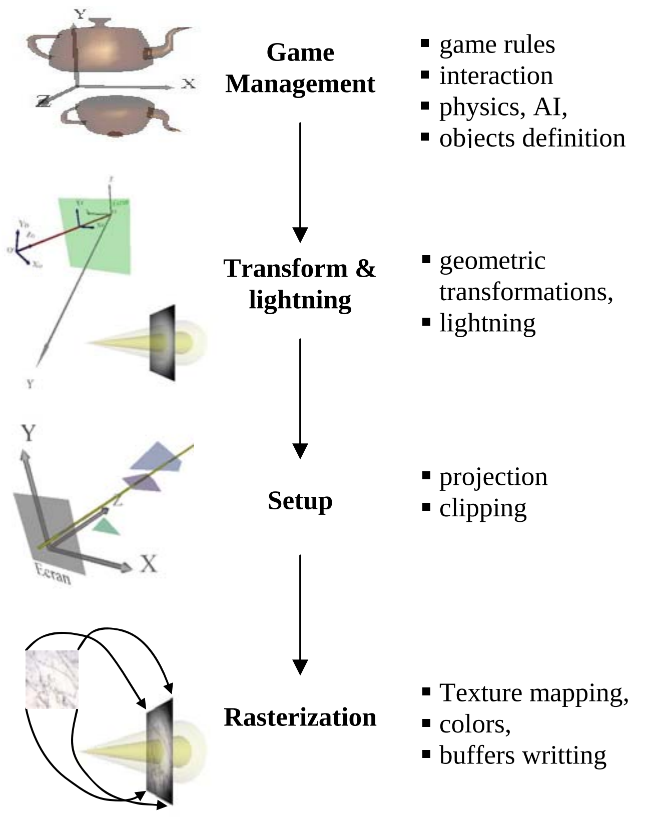


Fig. 2: Functions of the game engine.

To obtain a correct result, the engine must offer an optimal management of the scene in order to send to the renderer only the visible objects. It can also be useless to detect collisions between invisible objects (unless for sound effects). There are several strategies for organizing the scene in order to use culling techniques. The standard techniques are the following (Foley et al. 1997):

- hierarchical subdivision:
    - octrees
    - binary space partition
- occlusive sorting
    - portals (Lengyel 2002)

Transforming the scene graph into a culling structure (octrees or BSP) is made at the initialization stage. It takes into account only the static objects (i.e. the world, the landscape). To integrate dynamic elements of the scene would require recomputing the whole structure at each frame. That is not possible in real time. Hence, since the world in IIWI is entirely dynamic, it can only be culled by using bounding geometries which is the less efficient technique.

The network messaging system between players must be scalable to take into account high and low bandwidth connections. Messages must be short, compressed and regular. Algorithms like dead reckoning are also necessary to take care of the simulation between messages.

The apparent simplicity of IIWU's gameplay should not be misleading. It is much more complicated technically than one can suppose by reading its simple textual description: an arena filled with cubes and players. The characteristics are as follows:

- Each cube of the scene is an actor.
- Each cube can have its parameters changed.
- Each cube is subject to collisions.
- Each cube has its own position in space.
- Each cube can react differently under the effect of illumination.

There is at least a matrix of 50 by 50 cubes which composes a scene. Knowing that the arena, in the worst case, can be entirely visible, that makes us a total of 2500 active objects in the scene without counting the players who are at least 5. This raises many problems including:

- Numerous messages exchanges between clients to notify their position change or a cube state change.
- Physical simulation of many objects.
- Graphical rendering of many cubes highlighted by several light sources.

Those problems shouldn't be left without solutions. It seems however that no known game engine can process the illumination problem. They are made for specific uses which don't include multiple dynamic lightning. Hence a tailored game engine is necessary to remove these limitations.

GRAPHIC IMPLEMENTATION

One of the main problems with our game is the graphical design. It uses many lights and many simple structures (cubes) with few vertices. Finding the right algorithm that meets our needs was a hard work. Next, are different tries made to obtain the right luminous interaction on cubes.

Classical rendering

The lightning in OpenGL is made at the vertex level (Angel 2005, OpenGL 1999a, OpenGL 1999b). A luminous intensity is associated to each vertex and an interpolation is made to determine the color of every pixel of the projected face. Hence, to have nice smooth contours of a spotlight on an object requires the multiplication of the faces. Obviously, as we can see it in the figure 3, the Gouraud interpolation (Gouraud 1971) between vertices does not have the required visual effect.

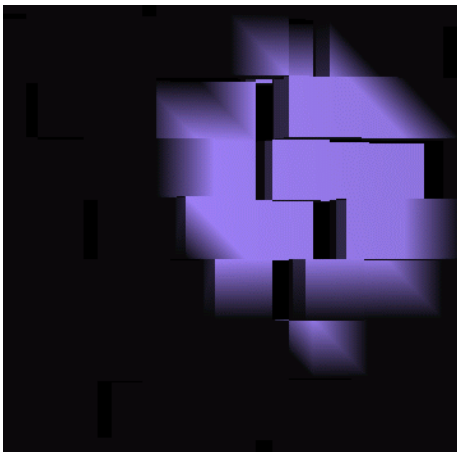


Fig. 3: Classical OpenGL illumination rendering.

Texture projection

OpenGL makes it possible to define textures associated with the faces (Astle 2001). 2D texture coordinates are used to map the image perpendicularly on the face (Lengyel 2002). OpenGL also introduces a third and a fourth coordinate to specify a perspective projection. It is possible to deform the texture according to the axis of projection (Reeves and Salesin 1987, Segal et al. 1992).

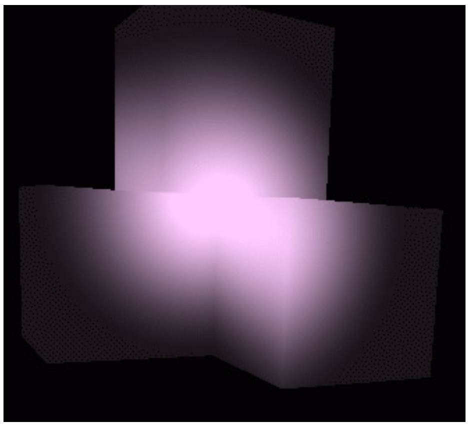


Fig. 4: Texture projection method.

We developed such a texture projection model. As one can see on figure 4, the result is definitely more interesting. The image of the spot can be defined in an algorithmic way or generated via a graphic editor.

Multitexturing, Blending and Multipass

We saw that it is possible to obtain an interesting spot rendering thanks to the projection of textures. In IIWU, the player is represented by the impact of the spotlight on the arena. Since the game is multi player, it will be necessary for us to project as much texture as visible players. This is a real problem because mixing spotlights handled by multiple players isn't that easy. We also have to mix those dynamic lightning textures with the cube texture.

Multi-texturing is an OpenGL extension since version 1.2. Even if this extension is supported in hardware on the recent graphics cards, the majority support only a finite number of cumulated textures (from 2 to 6 textures). It means that our application, even if it can take advantage of the hardware support of the multitexturing, can not manage multiple spotlights in one single pass. One can see in figure 5 how the luminous intensity of pixels increases where two textures (i.e. two spots) are superposed.

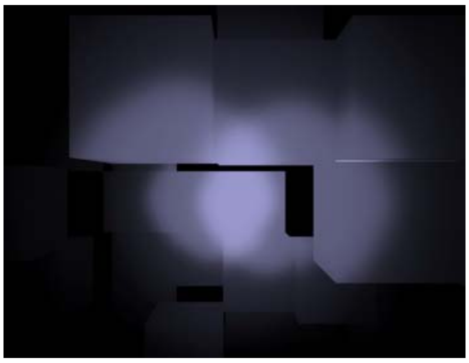


Fig. 5: Multitexturing and blending lightning method.

The result is rather good and this method can succeed with any kind of lighting images. A player can tune his spotlight like he/she would with a usual avatar. To draw more spotlights we have to make a multi-pass rendering. The scene will be drawn as many times as there are lights to mix (blending). It is obvious that if the scene has a great number of polygons, the frame rate risks falling under an acceptable limit.

Stencil shadow

The stencil buffer is usually used to render shadows (Everitt and Kilgard 2002). Our problem is the opposite: to render lights in a dark environment. We thought about how to use the stencil shadow to answer this particular problem: We are in the black and we want a luminous impact instead of a shadow. Hence, we thought about modifying the stencil shadow algorithm in order to take care of both spotlight and shadow.

By using the z-fail algorithm for the luminous cone we obtained the impact of the spots. Usually only few polygons are concerned while rendering in the stencil buffer. Hence, it was quite easy to project several spotlights.

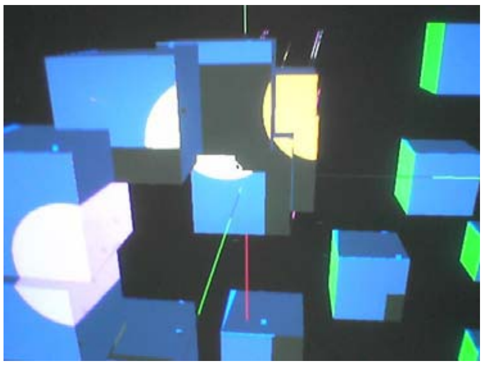


Fig. 6: Stencil and Z-fail algorithm method

Figure 6 gives an example of what can be achieved with this method. This rendering technique was abandoned because of the too great clearness of the spots and shadows edges. The same problem can also be seen in the very distinguishable shadows of Doom III. This technique was thus definitively put aside in aid of the projection of texture.

Shadow mapping

Considering it very schematically this technique consists in projecting the image of the scene from the point of view of the light source, colored in black. The algorithm takes two passes:
1. Compute the depth buffer from the point of view of the light source. One obtains a "depth map" or "shadow map".
2. Compute the scene from the camera's point of view using the map. For each rasterized fragment, one determines xyz position of the fragment according to the light. Then one compares the depth value of on the map (A) in xy with the position of the fragment in Z (B). If B>A the fragment is in a shadow because an object is closer to the light source. If A >= B, the fragment is enlightened.

The problem is that the depth map has a lower resolution than the frame buffer. Thus, more the light source is far, more the phenomenon of aliasing increases. A second problem is the multiple renderings needed; one for each light source and an additional one from the camera's point of view. The last disadvantage is the use of additional textures that slows down dramatically the rendering speed for multiple light sources.

CONCLUSION & FUTURE WORK

IIWU is an atypical game. It is from the conceptual point of view since its playing rules are completely original. It also is from the typological point of view since it doesn't enter in any category given in [REF]. And it is also atypical from its graphical point of view since it is based on cubes and lights. It requires a brand new game engine to take care of the graphics, network and scene management. Indeed, the management of several dynamic lights, the real time interaction with complex movements or an open arena are generally not allowed in today's game. Hence, these functionalities are not present in today's game engines.

The graphics problem remains open. The idea is to create an engine making it possible to manage as many dynamic lights as needed. The objective is to obtain a fast light and shadow projection algorithm. We shall now investigate solutions based on pixel shaders. Our idea is to use the z-fail algorithm within a fragment program with a higher definition. Another solution could be to use the stencil shadow technique and apply a blur on the edges of the projected lights.

REFERENCES

Angel E., *Interactive Computer Graphics : A Top-Down Approach using OpenGL*. Addison Wesley, New York, 2005.

Astle H., *OpenGL Game Programing*. Prima tech's, 2001.

Eberly D. H., *3D Game Engine Design A practical approch to real time computer graphics*. Morgan Kaufmann Publishers, 2000.

Everitt C., Kilgard M. J., "Practical and robust stenciled shadow volumes for hardware-accelerated rendering", Technical report, NVIDIA Cooperation, March 2002. Published online at developer.nvidia.com.

Foley J. D., Van Dam A., Feiner S. K., Hughes J. F., *Computer Graphics Principles and Practice 2nd ed in C*. Addison-Wesley, New York, 1997.

Gouraud H., "Continuous shading of curved surfaces", *Communications of the ACM*, 18, 6, 623–629, June 1971.

Lengyel E., *Mathematics for 3D Game Programming & Computer Graphics*. Charles River Media, Game Development Series, Hingham-Massachusetts, 2002

Moore M., Wilhelms J., "Collision detection and response for computer animation", *In Proc. Of SIGGRAPH'88*, volume 22, pp. 289-298, Aug. 1988.

Naylor B., Amatodes J.A., Thibault W., "Merging BSP trees yields polyhedral set operations", *In Proc of SIGGRAPH'90*, volume 24, pages 115-124, Dallas, Texas, USA, Aug. 1990.

Noborio H., Fukuda S., Arimoto S., "Fast interference check method using octree representation", *Advances Robotics*, 3(3), pp. 193-212, 1989.

OpenGL Architecture Review Board, *OpenGL Programming Guide*, 3rd edition, Addison-Wesley, 1999.

OpenGL Architecture Review Board, *OpenGL Reference Manual*, 3rd edition, Addison-Wesley, 1999.

Reeves W. T., Salesin D. H., Cook R. L. "Rendering antialiased shadows with depth maps", *In Proceedings of SIGGRAPH'87*, pages 283-291. ACM Press, 1987.

Segal M., Korobkin C., van Widenfelt R., Foran J., Haeberli P., "Fast shadows and lighting effects using texture Mapping", *In Proceedings of SIGGRAPH'92*, pages 249-252. ACM Press, 1992.