The first person perspective and ways of achieving it

(with an overview of Windows 95 Direct-Draw™)

P.S.Ranganathan T.N.C.Venkata Rangan

SRI VENKATESWARA
COLLEGE OF ENGINEERING,
PENNALUR.

It all started with Wolfenstein and DOOM ... *(the Coke of computing)*

Hard Facts

- ✓ Dedicated Web site (www.doom.com)
- ✓ many newsgroups
- ✓ ftp sites
- ✓ fan groups
- ✓ CD-ROMs with nothing but DOOM info and add-ons

The one thing Unix and Windows hackers agree on

- ✓ QNX
- ✓ OS/2
- ✓ SGI Irix v5.2
- ✓ LINUX
- ✓ MAC
- ✓ WINDOWS 95
- ✓ NEXTSTEP
- ✓ Sega 32X

Serious psychological and social research on the effects of DOOM is going on

Books have been published and the brand-name carries a lot of market value.

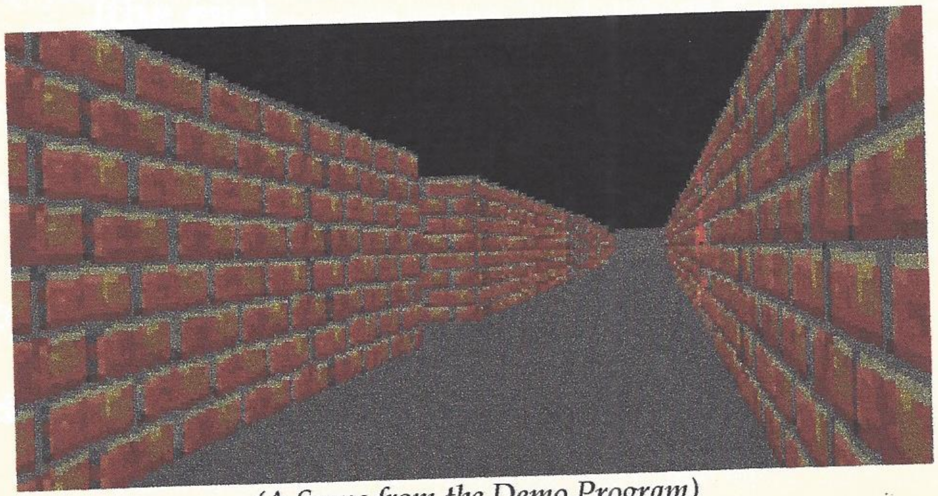
Looking back

- Prince of Persia is flat and 2D. Wolf is 3D
- Unlike Prince of Persia, in Wolf your eye is the camera
- Faster rendering

What is FPP?

A view on the screen that is close to what would be seen by the user, if he were standing in the viewpoint that he the program assumes him to be.

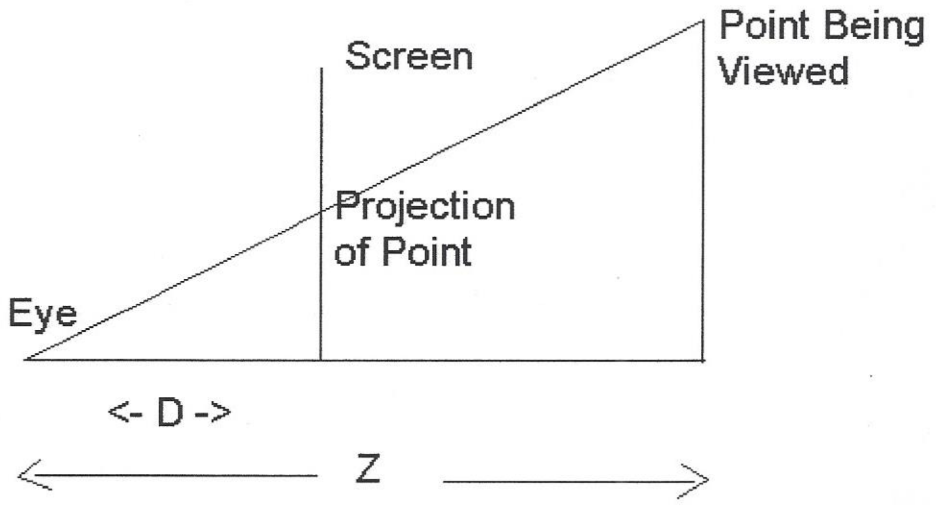
THE PROGRAM CREATES A VIRTUAL WORLD FOR THE USER.


(A Scene from the Demo Program)

3D graphics on a 2D display device

How we see - perspective projection

Perspective projection



✓A screen is between the viewer and object
✓All the light rays converge on a point
(the eye)
✓Objects far away appear smaller
(proportional to $1/z$)

How to render a 3D world?

A data structure is transformed into a 3D view

What it is?

The first person perspective(fpp) aims to provide the user with a view on the screen that closely approximates the scene that would be seen by the user, if he were standing in the viewpoint that he the program assumes him to be. That is the program tries to create a virtual world for the user. Wolfenstein 3D and DOOM are excellent examples of first person perspective.

Need for the first person perspective

Gone are the days when you were content to type your requests (not commands) to the computer and the computer took its own sweet time to respond. Now you want the info NOW!. With the changed user perspective, it is natural that interactivity and real-time response is expected of games. For example, the same game Tetris is available under both Windows 95 and Unix character based terminals. Which one would you prefer?

Applications of fpp

It is unimaginable to think of the game DOOM implemented in any other way. It would just lose its punch.

In *Disclosure* by Michael Crichton, we see an example of first person perspective in the scene where the database is searched by the person. This idea has been implemented in a software product. The results of a search are presented in 3D with closest matches in front. We see that fpp can be used creatively in other fields apart from games. Virtual reality is fpp in the extreme, where even sensory input is processed and total immersion is aimed at.

Techniques needed for fpp

- ⇒Ray tracing (seldom used)
- ⇒Ray tracing (dungeon type of games)
- ⇒Texture Mapping
    - ⇒Non perspective corrected
    - ⇒Perspective corrected
- ⇒Cylindrical Projection (not much seen)

How to do it?

Generally, the scene to be rendered can be broken down into polygons. If it has curves, then they are approximated by polygons. The problem is now to decide which polygons are visible given the scene, viewpoint and direction of view.

Ray tracing

There are many approaches to solving this problem. If scene realism is of paramount importance, ray tracing is the obvious choice. The disadvantage with ray tracing is that speed is very low.

The advantage is that all physical phenomena like reflection, refraction etc. can be reproduced perfectly. This technique can generate photographic quality images. This technique is very well known.

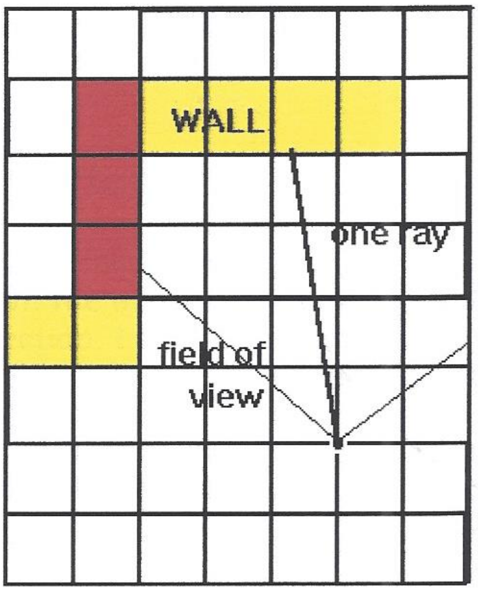
Ray Casting

Obviously this technique cannot be used for dynamically changing scenarios, like those is a game. We need a technique that is considerably faster than ray tracing. The improvement must be of the order of 2 magnitudes. A good solution is a poorer cousin of ray tracing called ray casting. This is not that well known. This method is very good for simulating dungeon type games. This is very fast but places quite a few limitations. This is the technique used in Wolf.

The idea is very simple. Consider a crossword like matrix. Assume this to be top view of the space we are going to render. If a square is shaded it is assumed to be a wall. All walls need not be of the same kind. We can have both stone walls and brick walls in the same view. To do this, we associate each square with a number giving the type of the wall. Squares with number zero are assumed to be air i.e. transparent.

Now come the assumptions (limitations). Since we are having squares, we cannot have walls turning at arbitrary angles. The angles must be 90^{o}. Also all walls are of equal thickness. Unless of course, two walls are placed touching each other. Another thing is that there is only the ceiling above the wall ie no lofts are allowed.

Let us consider how the scene is going to be rendered. Consider the scene given in the figure.



The things we need to know before rendering are the position of the viewer, direction of view and the field of vision. What we next do is to take the leftmost ray. We trace it (ie travel along it) until we meet a wall. When we meet a wall, we know that we have a wall, below it the floor and above the ceiling. Using the distance from the viewer, we can calculate the apparent height of the wall. Using this information we can paint one thin slice of the wall. This because there are no objects hiding the wall and wall extends to a certain fixed height, above which we definitely have the ceiling.

To provide textures like brick on the wall, we just change the slice that we paint. We make each square 64x64(say) and associate a bitmap with every face. By the intersection we know exactly which line of the face is visible. Using this information, the appropriate slice is extracted and displayed.

To fill the whole screen, we just change the position of the ray slightly and do the process again. We do this process as many times as there are pixels in the X direction. Suppose there are 320 pixels, we need to do this process 320 times for each frame. By contrast, we would be doing the inner loop 320*200 i.e. 64,000 times in ray tracing. This is the 2 order magnitude speed up I was talking about. One demo program I saw, hit 76 frames per second on a 486dx2 with a VESA SVGA card.

Due to the fixed bitmap resolution, we find that the image quality degrades as we go nearer to the wall. This is because as we go nearer, more than one ray intersects with the same slice of the wall, this results in distorted images. This can be clearly observed in Wolf and DOOM by going very close to the walls. The only cure for this is to digitize the textures at the resolution that is needed to display them when the viewer is very close.

Cylindrical Projection

The previously described projections use perspective projection to provide a depth cue. On the other hand, there is another projection called cylindrical projection. I have not seen it used in any product. This is not a "natural" projection. It can be understood in the following way. The scene to be viewed is assumed to be painted on a roll of paper. This roll of paper is pasted on a cylinder. The viewer is standing some distance away from the cylinder and viewing it. Panning of the image is done by rotating the cylinder.

(There was some talk about Microsoft and Apple standardizing this and bringing out tools to allow development of products using this technology. To digitize real world photos in this format we need a panoramic camera. This camera shoots on a continuous roll of film.)

Wolfenstein

Constraints on the world

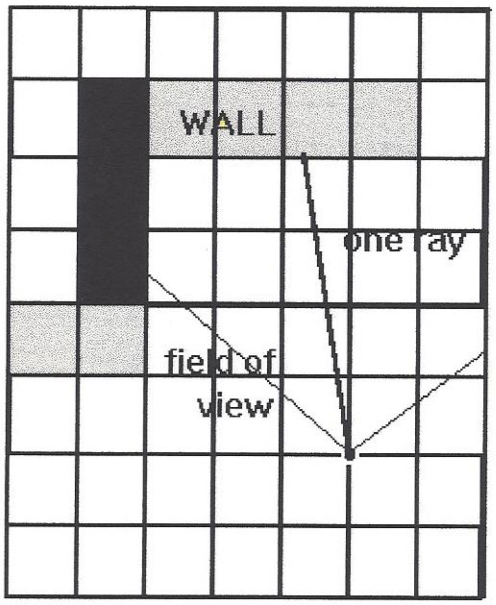
Orthogonal walls
Walls of the same thickness
Only one height level in the game

Ray Casting

It is a subset of ray tracing. It is a kind of ray tracing in which ray-object intersections are not pursued further.

Ray casting in Wolf

The data structure used is a 2D matrix(!!!) since the FPP is totally defined by the top view.



0	0	0	0	0	0	0
0	1	2	2	2	2	0
0	1	0	0	0	0	0
0	1	0	0	0	0	0
0	2	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

DOOM

Major differences from Wolf

Non orthogonal walls
Multiple levels in the (stairs etc.)

Where's the catch?

Only horizontal and vertical planes are used

How DOOM *may* be working

Clue is the 4 MB wad file

Stacks of 2D matrixes

Next approach : Pre computed view

✓What it is?
✓How DOOM's assumptions help
✓How to conserve memory

Hypothesis on how DOOM *may* be working

Must (can) we include this?

Basically uses ray casting. Consists not of squares but of units with width of 1. This will allow us to approximate diagonal lines quite well. Also has data saying for each unit how high it is. If it does not go up to ceiling, points to one above it. And so on.

1. Cast one ray flat on the ground. Print one slice for the height given in the data struct.

2. Cast another ray raised above the ground at height given in prev case. Print another slice.

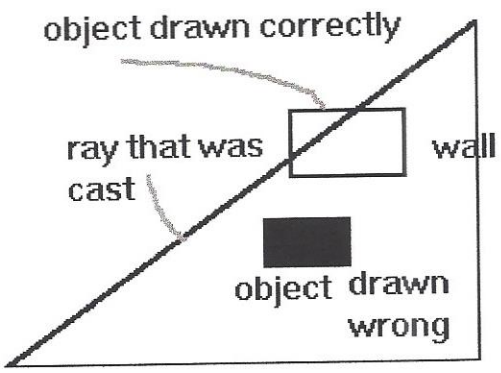
3. If necessary repeat step 2.

4. Print ceiling.

5. print floor.

6. Sweep ray a little right. Do the previous steps.

The catch in the algorithm is that it will fail to render cases like the following. When the flat ray is cast, it hits the wall. Next when the elevated ray is cast, the solid object is missed, when it should be actually drawn. The unfilled box is rendered correctly.



An optimization that is possible is that for tracing the elevated ray, a variation of Bresenham's algorithm in 3D can be used.
This probably needs polishing before it works right.

Windows 95 Direct Draw™

The Windows 95 Game SDK enables the creation of world class computer games. DirectDraw is a component of that SDK that allows direct manipulation of video display memory, hardware blters, hardware overlays, and page flipping. DirectDraw provides this functionality while maintaining compatibility with existing Windows 95 applications and device drivers.

The Windows Game Subsystem allows game authors an unprecedented level of access to the display and audio hardware while insulating them from the specific details of that hardware. The Windows Game Subsystem is built for speed. In keeping with these design goals, **DirectDraw is not a high level graphics API.**

DirectDraw for the Microsoft Windows operating system is a software interface which provides direct access to display devices while maintaining compatibility with Windows GDI. **It works with Windows 95 and will work with Windows NT.** DirectDraw provides a device-independent way for games and Windows subsystem software such as 3-D graphics packages or digital video codecs to access display device-dependent features.

DirectDraw works with a wide variety of display hardware, ranging from simple SVGAs to advanced hardware implementations providing clipping, stretching, and non-RGB color format support. The interface is designed so that **applications can request the capabilities of the underlying** hardware, then use those capabilities as required.

DirectDraw provides access to the following display device-dependent benefits:

- Support for double-buffered and page flipping graphics
- Access to, and control of, the video card's blitter
- Support for 3D 'z' buffers
- Hardware assisted overlays with z ordering
- Improved graphics quality through access to image-stretching hardware
- Simultaneous access to standard and enhanced display device memory areas

The DirectDraw HAL is hardware dependent and contains only hardware-specific code. The HAL implements only the device dependent code and performs no emulation.

The DirectDraw object represents the display device. There can be one DirectDraw object for every logical display device in operation. A game development environment, for instance, might have two monitors, one running the game using DirectDraw and one running the development environment using GDI.

A DirectDrawSurface object represents a linear region of display memory that can be directly accessed and manipulated. These addresses may point to visible frame buffer memory (primary surface) or to non-visible buffers (offscreen or overlay surfaces).

DirectDrawPalettes represent a 16 or 256 color-indexed palette. Palettes are provided for textures, offscreen surfaces, and overlay surfaces, all of which do not necessarily have the same palette as the primary surface.

Tomorrow's Video Techniques

- Overlays. Overlays will be supported so that page flipping will also be enabled within a window in graphic device interface (GDI). Page flipping is the double-buffer scheme used to display frames on the entire screen.
- Sprite engines, used to make overlaying sprites easier.
- Stretching with interpolation. Stretching a smaller frame to fit the entire screen can be an efficient way to conserve video RAM.
- Alpha blending, used to mix colors at the hardware pixel level.
- Three-dimensional (3-D) accelerators with perspective-correct textures. This feature will allow textures to be displayed on a 3-D surface so; for example, hallways in a castle generated by 3-D software can be textured with a brick wall bitmap that maintains the correct perspective.
- Z buffer-aware bit-block transfers for 3-D graphics.
- 2 megabytes (MB) of video memory is standard. 3-D games generally need at least this much video RAM.
- A compression standard so you can put more data into display memory. This standard will include transparency compression, be usable for textures, and be very fast when implemented in software as well as hardware.

Bibliography

1. Wolfenstein Documents.
2. DOOM FAQ's on the Internet.
3. Tricks of Game Gurus.
4. Windows 95 Games SDK Documentation.
5. Graphics Programming in C.