Release statement

This is a 100% handmade demo. Code was developed manually (with hundreds of hours of work), video backgrounds have been recorded with an actual camera, music has been recorded with a Kramer Floyd Rose Sustainer guitar and a BOSS SY-200 effect pedal, etc.

Just like in the olden days when there was no alternative.

This demo is unfinished, the engine is unfinished too. So this is just a techdemo introducing the current capabilities of the engine, which is:

- pure Assembly engine
- 320x200 movie background
- adaptive palette
- 8bit, 44100 Hz music
- 3D pointcloud rotation

I was in a rush, I had to release this demo now, because AI is coming. It is coming and it is growing fast. One or two years from now everyone can generate an engine like this in a moment. Future AI or LLM models can be used to solve super complex problems like developing DOS demos (or DOS Assembly engines) based on a prompt request.

Therefore I get ahead of things and I tried to think proactively. Eventhough I have been developing this engine in my freetime for many years now and I have been preparing a nice looking demo, I changed my mind and terminated the devlopment process. I decided to release the product as is, in order to demonstrate my product before AI becomes capable of doing demos (or demo engines) like this one (in a moment) based on a promt request. I simply released this product in order to prove that I was able to develop my own engine (on my own) with a tremendous amoun of hard work **without** the help of any AI.

So, it is 2024 now, and AI simply cannot do demos or demo engines like this yet. What about asking for parts of an engine? AI fails too.

Just for fun let's take a look at it and let’s see what AI responds when we're asking for an actual engine or its parts in 2024:

ChatGPT 4o

Generate a DOS, pure assembly demoscene demo engine for me, which is capable of playing 8bit, 44100 Hz music, rotates 3D pointclouds, the background is a movie, etc.
The demo engine should be ready to receive WAV files, background images and videos (for the background movie) and pointcloud files (in PLY format). If we organize the above mentioned resources in the correct order (in folders, etc.), the engine should create an actual DOS EXE from them so we can see these content running as a DOS demoscene product.

Creating a fully-fledged DOS-based assembly demoscene demo engine capable of rendering such features (8-bit, 44100 Hz music, 3D pointclouds, and background video) is a highly complex task. However, I can outline the approach, provide key assembly routines, and explain the steps to build this engine. Let's break it down:

---

1. Requirements and Tools

- **Assembler:** Use Turbo Assembler (TASM) or NASM.
- **Sound Library:** Implement or use a mixer for 8-bit PCM sound.
- **Graphics Mode:** Use Mode 13h (320x200, 256 colors) or higher for visuals.
- **Video Decoding:** Simplified custom playback mechanism for video backgrounds.
- **Pointcloud Visualization:** A 3D math engine for rotations and projections.
- **Resource Management:** Organize WAVs, images, and PLY files.

---

2. Resource Setup

Directory Structure:

```css
resources/
  music/
    track.wav
  video/
    background.raw (uncompressed frame-by-frame RGB data)
  models/
    pointcloud.ply
```

---

3. Core Engine Features

a) Sound Engine

Playback of 44100 Hz audio using the Sound Blaster DSP in pure assembly:

```asm
; Initialize Sound Blaster for 8-bit PCM
mov dx, 0x220  ; Base port of Sound Blaster
mov al, 0xD1   ; DSP reset
out dx, al
call delay     ; Small delay
in  al, dx     ; Acknowledge DSP
...
; Playback loop
```

b) Background Movie

Use pre-rendered raw frame data (320x200 or similar). Load each frame, write to the VGA buffer:

```asm
; Set VGA Mode 13h
mov ax, 0x13
int 0x10

; Load frame from file
mov dx, frame_buffer
mov es, dx
mov si, [frame_data] ; Source in memory
mov di, 0xA000       ; VGA memory start
rep movsb            ; Copy frame to video memory
```

c) Pointcloud Renderer

Load PLY, apply rotation matrix, and project onto 2D screen:

```asm
; Rotate point
fld   [angle]
fsincos
fstp  [cos_angle]
fstp  [sin_angle]

; Transform vertices
mov   ax, [x_coord]
imul  ax, [cos_angle]
mov   bx, [y_coord]
imul  bx, [sin_angle]
add   ax, bx
mov   [screen_x], ax

; Plot pixel
mov   ax, 0x0C00 ; Set pixel interrupt
mov   cx, [screen_x]
mov   dx, [screen_y]
int   0x10
```

d) Timing Control

Use the PIT for sync (vertical retrace or fixed intervals):

```asm
mov al, 0x36
out 0x43, al
mov ax, 11932
out 0x40, al ; 44100 Hz
```

---

4. Building the EXE

Bundling Resources into an EXE:

1. Write a resource packer (in C/ASM) that embeds WAV, raw video frames, and PLY data.
2. Engine loads resources dynamically from offsets in memory.

Makefile for Compilation:

```makefile
engine.exe: engine.asm resources.obj
    tasm engine.asm
    tlink /3 engine.obj
```

---

5. Sample Main Loop

```asm
start:
    call init_sound
    call load_video
    call load_pointcloud
    call main_loop

main_loop:
    call render_frame
    call play_audio
    jmp main_loop
```

---

6. Notes

- Implementing the full engine will require advanced knowledge of assembly and DOS architecture.
- Precompute video and organize resources for efficiency.
- 3D math can be optimized with fixed-point arithmetic.

Would you like code for any specific feature in more detail or guidance on setting up the environment?

Now let's give it a try and ask the AI to develop at least the 3D point rotating part, but instead of a complex task like rotating a pointcloud, ask the AI to rotate only a cube.

ChatGPT 4o

write an assembly (NASM, DOS) program that rotates a cube on the 320x200 graphical screen (8 points/edges representing the cube). make sure that the cube has a 256 color palette and also make sure that scene doesn't blink, meaning that program properly changes active and passive graphical pages

Here is an example of a NASM program that rotates a 3D cube on the 320x200 graphics screen in mode 13h (256-color VGA mode). The program includes double-buffering to eliminate screen flicker by drawing to an off-screen buffer and then swapping buffers.

Assembly Code (NASM, DOS):

```asm
; Cube rotation in 320x200 VGA mode with double buffering
; Assembled using NASM

org 0x100 ; COM program

section .data
vertices dw -50, -50, -50,   50, -50, -50,   50,  50, -50,  -50,  50, -50
         dw -50, -50,  50,   50, -50,  50,   50,  50,  50,  -50,  50,  50

edges    db 0, 1,  1, 2,  2, 3,  3, 0
         db 4, 5,  5, 6,  6, 7,  7, 4
         db 0, 4,  1, 5,  2, 6,  3, 7

palette  times 256 db 0 ; Palette storage (filled later)

section .bss
zbuffer  resb 320*200 ; Off-screen buffer
angle_x  resw 1
angle_y  resw 1
angle_z  resw 1

section .text
start:
    ; Set 13h graphics mode
    mov ax, 0x13
    int 0x10

    ; Initialize variables
    xor ax, ax
    mov [angle_x], ax
    mov [angle_y], ax
    mov [angle_z], ax

main_loop:
    ; Rotate and project cube
    call clear_buffer
    call rotate_cube
    call draw_cube

    ; Swap buffers
    mov ax, 0xA000
    mov es, ax
    mov si, zbuffer
    xor di, di
    mov cx, 320*200/2
    rep movsw

    ; Update angles
    add word [angle_x], 1
    add word [angle_y], 2
    add word [angle_z], 3

    ; Check for ESC key press
    mov ah, 1
    int 0x16
    jz main_loop
    mov ah, 0
    int 0x16
    cmp al, 0x1B
    jne main_loop

exit:
    ; Restore text mode
    mov ax, 0x3
    int 0x10
    ret

clear_buffer:
    ; Clear the off-screen buffer
    mov ax, 0
    mov di, zbuffer
    mov cx, 320*200/2
    rep stosw
    ret

rotate_cube:
    ; Rotate the cube in 3D space
    ; Uses simple fixed-point math
    mov ax, [angle_x]
    call sin_cos
    mov si, cx ; cos(angle_x)
    mov di, dx ; sin(angle_x)
```

```
    ; (similar rotation code for y and z axes)

    ret

draw_cube:
    ; Draw edges of the cube onto the off-screen buffer
    ; (Includes 3D to 2D projection and line-drawing code)
    ; Use the vertices and edges arrays
    ; Apply perspective projection: x' = x/z, y' = y/z
    ; Draw lines for each edge
    ret

sin_cos:
    ; Calculate sine and cosine of AX (angle in degrees)
    ; (simplified example using precomputed table or algorithm)
    ; Returns CX = cos(angle), DX = sin(angle)
    ret
```

Features:

1. **Double Buffering:**
   - The off-screen buffer `zbuffer` is used to render the cube before copying it to video memory (`0xA000`), preventing flickering.
2. **3D Rotation:**
   - The cube rotates around the X, Y, and Z axes using simple trigonometric functions (`sin_cos` procedure).
3. **Line Drawing:**
   - Cube edges are drawn based on the `edges` array. The `draw_cube` procedure includes Bresenham's line algorithm or similar logic to render edges.
4. **Keyboard Exit:**
   - Press `ESC` to exit the program and return to text mode.

Notes:

1. This program uses fixed-point arithmetic for efficiency since the DOS environment lacks floating-point support by default.
2. The sine and cosine functions can be implemented using a lookup table or approximations.
3. The palette setup is omitted for simplicity, but you can initialize a gradient palette by writing to VGA ports `0x3C8` and `0x3C9` .

Let me know if you'd like further details on specific sections!

It seems that it provided some parts, but the most important parts are missing (no sin/cos part, no palette, no 3D->2D projection, etc.):

```
draw_cube:
    ; Draw edges of the cube onto the off-screen buffer
    ; (Includes 3D to 2D projection and line-drawing code)
    ; Use the vertices and edges arrays
    ; Apply perspective projection: x' = x/z, y' = y/z
    ; Draw lines for each edge
    ret
```

Let's be fair and give it another try:

ChatGPT 4o

write a dos, nasm assembly code that rotates a cube (8 points representing the edges of the cube) on a 320x200 graphical screen. use FPU commands for sin or cos calculations

2/2

Below is a basic example of a DOS assembly program in NASM that rotates a 3D cube on a 320x200 graphical screen. It uses the x86 FPU for trigonometric calculations to determine the rotation of the cube.

Program Code

asm

Copy code

```
; Rotate a cube in 320x200 mode using NASM
; Assemble with: nasm -f bin -o cube.com cube.asm
; Run in DOSBox or a similar emulator.

org 0x100

section .data
; Define the 3D points of the cube (x, y, z)
cube_points dq -50, -50, -50    ; Point 1
            dq  50, -50, -50    ; Point 2
            dq  50,  50, -50    ; Point 3
            dq -50,  50, -50    ; Point 4
            dq -50, -50,  50    ; Point 5
            dq  50, -50,  50    ; Point 6
            dq  50,  50,  50    ; Point 7
            dq -50,  50,  50    ; Point 8

; Projection scale and screen offset
scale dw 100
screen_center_x dw 160
screen_center_y dw 100

; Rotation angles (in radians)
angle_x dq 0.0
angle_y dq 0.0
angle_z dq 0.0

section .bss
rotated_points resd 24    ; 8 points * 3 coordinates (x, y, z)

section .text
start:
    ; Set video mode 13h (320x200 with 256 colors)
    mov ah, 0x00
    mov al, 0x13
    int 0x10

main_loop:
    ; Clear screen (fill with black)
    xor di, di
    mov cx, 320*200/4
    xor eax, eax
    rep stosd

    ; Rotate cube
    call rotate_cube

    ; Draw cube
    call draw_cube

    ; Update rotation angles
    fld dword [angle_x]
    fldpi
    fadd dword [0.02]    ; Increment X angle
    fstp dword [angle_x]

    fld dword [angle_y]
    fldpi
    fadd dword [0.03]    ; Increment Y angle
    fstp dword [angle_y]

    ; Delay loop (cheap way to control speed)
    mov cx, 0xFFFF
.delay_loop:
    loop .delay_loop

    jmp main_loop

rotate_cube:
    ; Rotate each point and project to 2D
    lea si, [cube_points]
    lea di, [rotated_points]
    mov cx, 8 ; 8 points

    .rotate_point:
        ; Load the point coordinates
        fld qword [si]      ; Z
        fld qword [si+8]    ; Y
        fld qword
```

asm

```asm
        fld qword [si+16]   ; X

        ; Apply rotation around X-axis
        fld dword [angle_x]
        fsincos
        ; cos(angle_x) -> st(0), sin(angle_x) -> st(1)
        fxch st(1)
        fmul st(0), st(2)    ; sin(angle_x) * Z
        fld st(3)
        fmul st(0), st(1)    ; cos(angle_x) * Y
        fsubp st(1), st(0)   ; Y' = cos(angle_x)*Y - sin(angle_x)*Z
        fst qword [di+8]     ; Store rotated Y

        ; Apply rotation around Y-axis
        fld dword [angle_y]
        fsincos
        fxch st(1)
        fmul st(0), st(2)    ; sin(angle_y) * X
        fld st(3)
        fmul st(0), st(1)    ; cos(angle_y) * Z
        faddp st(1), st(0)   ; Z' = cos(angle_y)*Z + sin(angle_y)*X
        fst qword [di+16]    ; Store rotated Z

        ; Apply rotation around Z-axis
        fld dword [angle_z]
        fsincos
        fxch st(1)
        fmul st(0), st(2)    ; sin(angle_z) * X
        fld st(3)
        fmul st(0), st(1)    ; cos(angle_z) * Y
        faddp st(1), st(0)   ; X' = cos(angle_z)*X + sin(angle_z)*Y
        fst qword [di]       ; Store rotated X

        ; Project 3D points to 2D screen space
        fld dword [scale]
        fmul st(0), st(2)    ; scale * X'
        fadd dword [screen_center_x]
        fstp qword [di]

        fld dword [scale]
        fmul st(0), st(1)    ; scale * Y'
        fadd dword [screen_center_y]
        fstp qword [di+8]

        ; Move to next point
        add si, 24
        add di, 24
        loop .rotate_point

        ret

draw_cube:
    ; Draw lines connecting cube edges
    mov si, rotated_points
    ; TODO: Add drawing logic here (e.g., Bresenham's line algorithm)
    ret

exit:
    ; Return to text mode
    mov ah, 0x00
    mov al, 0x03
    int 0x10

    ; Exit program
    mov ax, 0x4C00
    int 0x21
```

Notes:

1. **Rotation Logic**: Uses the FPU `fsincos` to calculate sine and cosine of the angles for rotation.
2. **Projection**: Scales the 3D coordinates and maps them to the 2D screen.
3. **Graphics**: Implement a line-drawing algorithm (like Bresenham's) in the `draw_cube` procedure to connect the cube's vertices.
4. **DOS Environment**: Assemble with NASM and run using DOSBox or a similar DOS emulator.
5. **Optimization**: This is a basic implementation. Further optimization can be done for speed and clarity.

Let me know if you'd like assistance with specific parts, like implementing the line-drawing function or optimizing the rotation logic!

the code generated by the AI is just doesn't execute:

```
g:\deleteme\[redacted]\test>nasmw -f bin test.asm -o test.com
test.asm:54: expression syntax error
test.asm:59: expression syntax error
test.asm:79: expression syntax error
test.asm:86: comma or end of line expected
test.asm:87: comma or end of line expected
test.asm:87: comma or end of line expected
test.asm:88: comma or end of line expected
test.asm:89: comma or end of line expected
test.asm:89: comma or end of line expected
test.asm:90: comma or end of line expected
test.asm:90: comma or end of line expected
test.asm:96: comma or end of line expected
test.asm:97: comma or end of line expected
test.asm:97: comma or end of line expected
test.asm:98: comma or end of line expected
test.asm:99: comma or end of line expected
test.asm:99: comma or end of line expected
test.asm:100: comma or end of line expected
test.asm:100: comma or end of line expected
test.asm:106: comma or end of line expected
test.asm:107: comma or end of line expected
test.asm:107: comma or end of line expected
test.asm:108: comma or end of line expected
test.asm:109: comma or end of line expected
test.asm:109: comma or end of line expected
test.asm:110: comma or end of line expected
test.asm:110: comma or end of line expected
test.asm:115: comma or end of line expected
test.asm:115: comma or end of line expected
test.asm:120: comma or end of line expected
test.asm:120: comma or end of line expected
```

If we go ahead and fix above listed errors, then other errors occur.

And on top of that the main part is missing again:

```
draw_cube:
    ; Draw lines connecting cube edges
    mov si, rotated_points
    ; TODO: Add drawing logic here (e.g., Bresenham's line algorithm)
    ret
```

Final conclusion

In 2024 AI tries its best, but it fails. Seemingly it doesn't understand Assembly in general. AI cannot be used to implement complex problems written in Assembly yet.