Doom Engine Modification

*-- Yufan Lu, Bhumitra Nagar*

Purpose

In this article, we’ll present how to find the code points where we can modify the engine to be:

1. Free look
2. Accurate aiming
3. Third personal camera

Actually, these features have already existed in the code, nevertheless, Hidden deeply.

Compiling ZDoom Source Code

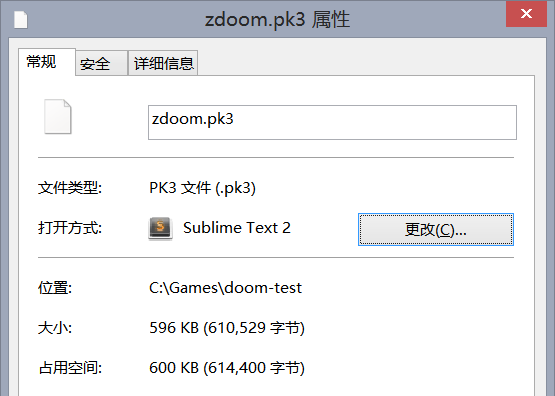
Operating System: Windows 8.1

IDE: Visual Studio 2013

Download: <http://zdoom.org/Download>

Required Libraries: Windows SDK for Windows 8.1, DirectX SDK, [FMOD](http://www.fmod.org/download/fmodex/api/Win/fmodapi42631win32-installer.exe) Ex

1. Install the required libraries listed above.
2. Open the zdoom’s project properties, add the FMOD’s include and lib path into the path.
3. Build the project.
4. Make sure you have the zdoom2.pk3 and DOOM2.wad, place them where the zdoomd.exe is.



Double check the zdoom.pk3 is 596KB version, because when we run the game on Bhumitra’s computer, and he has a 476KB version of zdoom.pk3, which resulted the failure of game execution.

1. Run the game.

Free look

Ok, the original DOOM doesn’t allow you to look up and down, when you move your mouse up and down, the character will be moving forward/backward instead of looking up/down.

Can we modify it to look up and down?



(Pic 1. Aiming at the ceiling)

Code snippet:

|  |
| --- |
| ***// D\_main.cpp, line 322, D\_PostEvent function***  int look = int(ev->y \* m\_pitch \* mouse\_sensitivity \* 16.0);  if (invertmouse)  look = -look;  G\_AddViewPitch (look);  events[eventhead].y = 0; |

What this snippet does is:

1. Calculate the movement of mouse along in y axis.
2. Invert it if the invertmouse is selected.
3. Use the function G\_AddViewPitch to pitch the camera
4. Set the y movement to be zero, so the character won’t be moving forward or backward.

Accurate aiming

Now we have the free look, but in original DOOM, the auto aim mechanics exists, which will compensate the shortage that you can’t look up and down. So we need to cancel this, and make it an accurate aiming.



(Pic 2. Shooting the position underneath the enemy but the enemy takes the damage)

Code snippets:

|  |
| --- |
| ***// p\_map.cpp, line 3369, P\_AimLineAttack function***  if (t1->player != NULL)  {  aim.shootz += FixedMul (t1->player->mo->AttackZOffset, t1->player->crouchfactor);  }  else  {  aim.shootz += 8\*FRACUNIT;  }  ***// p\_map.cpp, line 3405, P\_AimLineAttack function***  aim.toppitch = t1->pitch - vrange;  aim.bottompitch = t1->pitch + vrange; |

These code snippets makes the shooting offset, thus we just comment them, and then the auto aiming is canceled.



(Pic 3. Shooting underneath and the enemy does not take the damage)

Third personal camera

Actually, the original DOOM has the third personal camera implemented as a cheating mode. So we’ll dig out where these codes are.

|  |
| --- |
| ***// r\_utility.cpp, line 776, R\_SetupFrame function***  if (player != NULL && gamestate != GS\_TITLELEVEL &&  ((player->cheats & CF\_CHASECAM) || (r\_deathcamera && camera->health <= 0)))  {  // [RH] Use chasecam view  P\_AimCamera (camera, iview->nviewx, iview->nviewy, iview->nviewz, viewsector);  r\_showviewer = true;  }  ***// p\_map.cpp, line 4124***  void P\_AimCamera (AActor \*t1, fixed\_t &CameraX, fixed\_t &CameraY, fixed\_t &CameraZ, sector\_t \*&CameraSector)  {  fixed\_t distance = (fixed\_t)(chase\_dist \* FRACUNIT);  angle\_t angle = (t1->angle - ANG180) >> ANGLETOFINESHIFT;  angle\_t pitch = (angle\_t)(t1->pitch) >> ANGLETOFINESHIFT;  FTraceResults trace;  fixed\_t vx, vy, vz, sz;  vx = FixedMul (finecosine[pitch], finecosine[angle]);  vy = FixedMul (finecosine[pitch], finesine[angle]);  vz = finesine[pitch];  sz = t1->z - t1->floorclip + t1->height + (fixed\_t)(chase\_height \* FRACUNIT);  if (Trace (t1->x, t1->y, sz, t1->Sector,  vx, vy, vz, distance, 0, 0, NULL, trace) &&  trace.Distance > 10\*FRACUNIT)  {  // Position camera slightly in front of hit thing  fixed\_t dist = trace.Distance - 5\*FRACUNIT;  CameraX = t1->x + FixedMul (vx, dist);  CameraY = t1->y + FixedMul (vy, dist);  CameraZ = sz + FixedMul (vz, dist);  }  else  {  CameraX = trace.X;  CameraY = trace.Y;  CameraZ = trace.Z;  }  CameraSector = trace.Sector;  } |

In the first snippet, we found player->cheats & CF\_CHASECAM, it’s to test if the current player’s cheating status, if it’s in CF\_CHASECAM mode (chase camera), then execute the following code. That’s it, the third personal camera mode. Then we make it true…



That’s cool… Wait, what’s that hand with the gun?

I think it’s the game developer separated the drawing process, they drew the 3D space, then the hand decay, so we need to disable it.

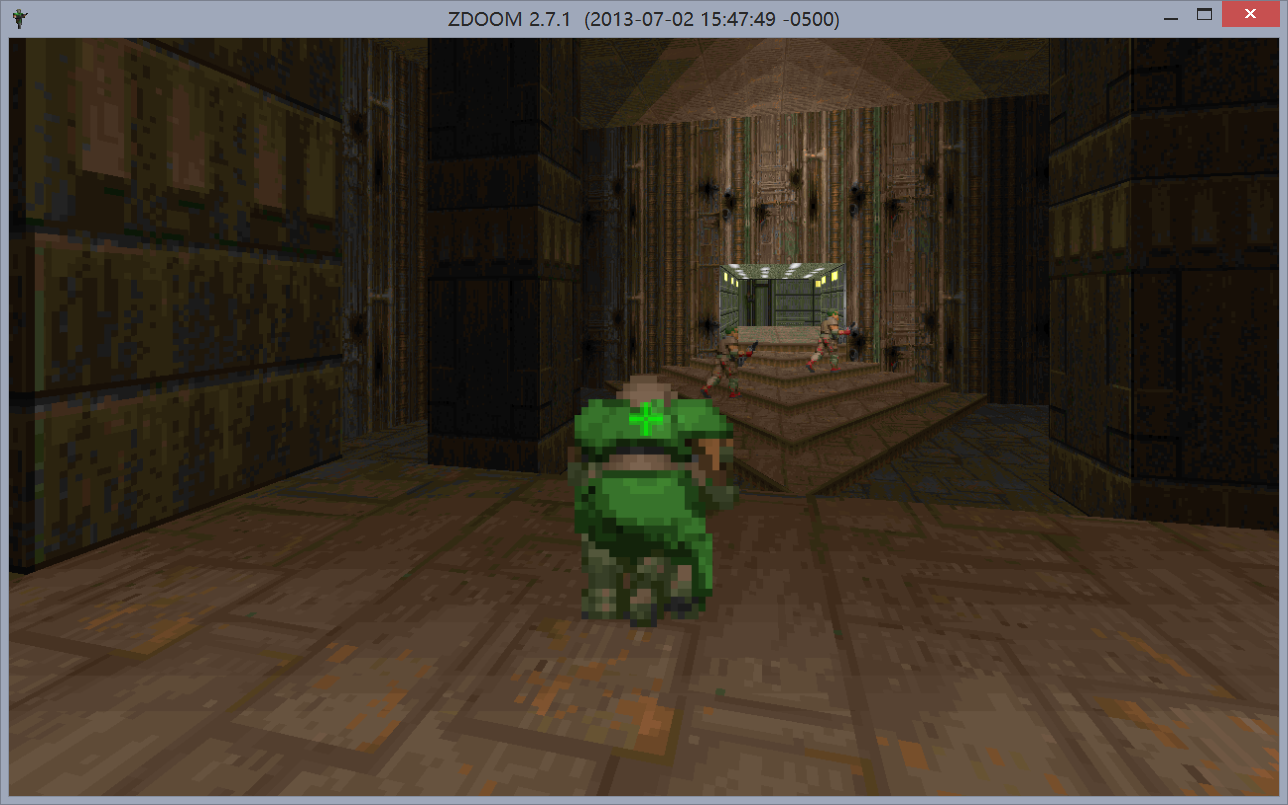
|  |
| --- |
| ***// r\_things.cpp, line 1166***  void R\_DrawPlayerSprites ()  {  int i;  int lightnum;  pspdef\_t\* psp;  sector\_t\* sec = NULL;  static sector\_t tempsec;  int floorlight, ceilinglight;  F3DFloor \*rover;  if (!r\_drawplayersprites ||  !camera->player ||  (players[consoleplayer].cheats & CF\_CHASECAM))  return;  if(fixedlightlev < 0 && viewsector->e && viewsector->e->XFloor.lightlist.Size()) {  for(i = viewsector->e->XFloor.lightlist.Size() - 1; i >= 0; i--)  if(viewz <= viewsector->e->XFloor.lightlist[i].plane.Zat0()) {  rover = viewsector->e->XFloor.lightlist[i].caster;  if(rover) {  if(rover->flags & FF\_DOUBLESHADOW && viewz <= rover->bottom.plane->Zat0())  break;  sec = rover->model;  if(rover->flags & FF\_FADEWALLS)  basecolormap = sec->ColorMap;  else  basecolormap = viewsector->e->XFloor.lightlist[i].extra\_colormap;  }  break;  }  if(!sec) {  // …… |

We found:

|  |
| --- |
| if (!r\_drawplayersprites ||  !camera->player ||  (players[consoleplayer].cheats & CF\_CHASECAM))  return; |

If the player’s cheating mode has the CF\_CHASECAM flag turned on, it’ll return the function directly…

So we return the function directly. It’s done.



Conclusion