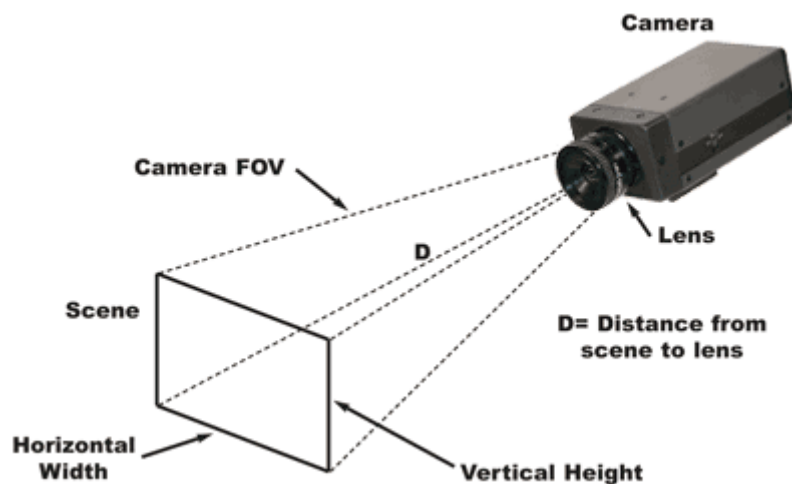


Camera

Camera is moving (rotating) left and right, eventually stopping on each end point and waiting specified amount of time (pause). During rotation, camera is detecting objects which appears inside camera's visibility area (called *camera zone*). Default algorithm for detecting objects visible by camera lens (default GSC script) is not working well in Stalker as it is possible that camera can spot objects hiding behind camera (how the f*ck is that possible is beyond me, I have tried everything and it's still buggy ...). Algorithm is based on creating a scene (*camera zone*) where camera can see objects. This is done by calculating FOV angles and distance (D) from lens, so basically any object inside scene built by FOV angles and in certain distance from camera lens is visible by camera. http://en.wikipedia.org/wiki/Angle_of_view



To fix enemy detection, I've added a space restrictor (camera zone) bound to camera. So algorithm works like that: if enemy is inside any camera zone bound to camera (can be more then one) then script builds camera scene (FOV, object – lens distance) and if object is inside camera scene then it can be spotted. This isn't a perfect solution, however at least it works and besides, using camera zones it is easier to specify where camera can spot enemies (useful for places with many obstacles).

Project files

- sounds\camera\beep.ogg – beeping sound if camera spot an enemy
- sounds\camera\move.ogg – rotation sound
- meshes\equipment\camera_cover.ogf – hack for hit callbacks (read below)
- config\scripts\camera.ltx – camera logic sections (*)
- config\scripts\camera_cover.ltx – wooden box logic (for hit callback hack) (*)
- spawns\all.spawn – spawn sections for camera and camera zone

- config\camera.ltx – sections for camera and camera zone (**)
- scripts: ph_camera, bind_camera_zone, modules, xr_effects

** logic can be defined in all.spawn, although it is easier to put logic in external files – we don't have to recompile all.spawn each time we edit camera's logic*

*** use level editor instead – menu spawns\lost alpha\camera for camera and spawns\lost alpha\camera_zone for camera zone*

Spawning

Camera has to be spawned as **camera** section (O_SEARCH class, same as projector) so it can rotate left – right and up – down. To spawn camera, select **spawns\lost alpha\camera** and edit logic – preferably specify a path to ltx file with camera logic (external ltx file is easier to edit later on – no need to recompile all.spawn). Be sure to change camera name to some **unique** name. This is very important because of hack for camera hit callbacks. Here's an example of spawned camera (important parts are highlighted, camera.ltx with camera logic is described in section [camera logic settings](#)):

```
[852]
; cse_abstract properties
section_name = camera
name = temp_camera
position = -146.399993896484,-27.5,-363
direction = 0,89.5,0

; cse_alife_object properties
game_vertex_id = 31
distance = 0
level_vertex_id = 86409
object_flags = 0xffffffff3a
custom_data = <<END
[logic]
cfg = scripts\camera.ltx
END

; cse_visual properties
visual_name = equipments\camera
```

Then spawn *camera cover* – this is a hack for camera hit callbacks. Basically, if we spawn camera as O_SEARCH class object, it can rotate but cannot receive hit/death callbacks. If we spawn camera as physic_destroyable_object it can receive callbacks but cannot rotate. To fix this we use *camera cover* spawned as physic_destroyable_object. This is a wooden crate (box) with invisible shadders which covers the whole camera. Idea is this: if user is shooting towards camera, all the bullets will hit that wooden box which then will receive hit callback. Using this callback camera will be deactivated (explodes). This is why camera **must have unique name** because its name has to be used in *camera cover's* logic.

Here's an example of *camera cover* (important parts are highlighted):

```
[853]

; cse_abstract properties
section_name = physic_destroyable_object
name = tmp_box_on_camera
position = -147.399993896484,-27.2000007629395,-363
direction = 1.60000002384186,0,1.60000002384186

; cse_alife_object properties
game_vertex_id = 31
distance = 0
level_vertex_id = 86409
object_flags = 0xffffffff3a
custom_data = <<END
[logic]
cfg = scripts\camera_cover.ltx
END

; cse_visual properties
visual_name = equipments\camera_cover

; cse_ph_skeleton properties
skeleton_flags = 1

; cse_alife_object_physic properties
physic_type = 0x3
mass = 10
fixed_bones = link
```

Just make sure that you spawn *camera cover* as *physic_destroyable_object* (so it can receive hit/death callbacks) and that camera is covered by this *camera cover* box, also that it has *fixed_bones* specified (so it can float in the air).

Now take a look at *camera cover* logic:

```
[logic]
active = ph_idle
on_hit = hit
```

```
[hit]  
on_info = %=camera_got_hit(temp_camera)%
```

```
[ph_idle]
```

So when *camera cover* receive hit callback, function `camera_got_hit()` will be called. As an argument, specify camera name (this is why camera name has to be unique). We can eventually use story ids if using names is problematic (just let me know).

Now, since we have spawned camera and *camera cover* it's time to spawn *camera zones* and add logic for camera. First let's handle *camera zones*. Those are simple space restrictors, so you can use any shape you see fit to define zones where camera can see enemies. You can even define several *camera zones* for one camera. Just remember that *camera zone* names **must be unique** because they are needed in camera logic.



To spawn *camera zone* select [spawns\lost alpha\camera_zone](#). Here's an example of *camera zone* (important parts are highlighted):

```
[8625]  
; cse_abstract properties  
section_name = camera_zone  
name = esc_cam_zone_01  
position = -152.985061645508,-29.9456653594971,-363.080261230469  
direction = 0,0,0  
; cse_alife_object properties  
game_vertex_id = 33  
distance = 0
```

```

level_vertex_id = 96336
object_flags = 0xffffffff3e
; cse_shape properties
shapes = shape0
shape0:type = sphere
shape0:offset = 0,0,0
shape0:radius = 4
; cse_alife_space_restricter properties
restricter_type = 3

```

Camera logic settings

This settings defines camera behavior:

- **state** – camera state, can be either **off** (camera is disabled or destroyed) or **path_tracking** (rotating); other states are evaluated only via script (such as enemy tracking, waiting on end points, etc). If you set **state** to **path_tracking**, you must specify a way point for camera which defines at least 2 points (end points) between which camera will be rotating – typically one point on left camera's side and another on right. You can define more points (middle points) at which camera will be *looking at* while rotating – this depends on how wide *camera zone* is,
- **path_look** – way point name consisting of at least 2 points (end points) between which camera will be rotating. This is needed only if camera **state** is set to **path_tracking**. Way point is not requiring any flags or signals, however do not place way point too close to camera – that would force camera to *look* down underneath which can look odd. Here's an example of way point with 2 end points (left, right):

```

[esc_camera_look_2points]
points = p0,p1
p0:name = name00
p0:position = -148.761138916016,-30.1068019866943,-358.168518066406
p0:game_vertex_id = 17
p0:level_vertex_id = 100794
p0:links = p1(1)

p1:name = name02
p1:position = -147.65705871582,-29.7977733612061,-371.756530761719
p1:game_vertex_id = 15
p1:level_vertex_id = 102976

```

- **start_wp** – number (index) of first way point at which camera will look; by default camera will look first at point 0; this is needed only if camera state is set to **path_tracking**
- **stop_at_end_points** – this is Boolean value (either **true** or **false**) which decides whether camera will stop and wait at end points (both – left and right). This gives nice effect that camera is paused for some time before rotating back to previous point. Needed only if camera **state** is set to **path_tracking**; by default it is set to **true**, so camera will stop and wait at end points; pause time is set to ca 3 seconds
- **speed_h** – horizontal rotation speed; this decides how fast camera rotates in left – right direction; It goes actually pretty fast, so value should be lower then 0.5 – for instance value 0.2 (which is default) is good enough
- **speed_v** – vertical rotation speed; this decides how fast camera rotates in up – down direction; Just like with horizontal speed, value should be lower then 0.5 – again, value 0.2 (default) seems to be enough
- **fov_angle_min** – minimum FOV angle which depends on place where camera was spawned and is used for enemy detection; rather needs to be set with trial & error approach; default value is 5
- **fov_angle_max** – same as above, except this defines maximum FOV angle; default value is 30
- **fov_zones** – comma separated list of *camera zones* names – space restrictors where enemies can be spotted; if you won't specify any *camera zones* then enemies will not be tracked at all
- **track_mutants** – Boolean value which decides whether camera will track mutants which enter camera's *camera zones* specified in **fov_zones** field
- **enemies** – comma separated list of communities to track; for instance you can set camera enemies to **bandit** and camera will track only bandits; if you want camera to track user, just add **actor** as enemies

If *camera cover* received hit (actor shoot camera) then signal **camera_destroyed** is issued and it should be used to switch scheme section (camera goes to **off** state). However, it is better to exit logic: `on_signal = camera_destroyed | nil` – destroyed camera it not usable anymore so no need to idle in **off** state (unless we decide that in certain situations camera should be “repaired” after some time and becomes usable again – this can be implemented fairly easy, just give me a shout and I'll alter camera script, but for now, exit from scheme when signal was triggered).

Here's an example of camera logic:

```
[logic]
active = ph_camera@on

[ph_camera@on]
state = path_tracking
path_look = esc_camera_look_2points
start_wp = 0
speed_h = 0.2
speed_p = 0.2
fov_angle_min = 5
fov_angle_max = 30
fov_zones = esc_cam_zone_01
enemies = actor
track_mutants = true
stop_at_end_points = true
on_signal = camera_destroyed | nil ;--ph_camera@off

[ph_camera@off]
state = off
```

barin, 2010-06-26