

# GUIDE TO VEHICLE INSTRUMENTS IN OFP

## 1. Overview

This document describes cockpit instruments in **Operation Flashpoint / ArmA: Cold War Assault**. What types there are, how they're made and how they function. It's a guide for modders who create their own vehicles.

The game engine supports the following dynamic on-board mechanisms:

- dashboard light
- damage alarm
- 10 different gauges
- head-up display

I'll go through each one using original vehicles as examples. I assume you're already familiar with addon making and modelling. If not then see [tutorials](#).

Creative modders created custom instruments with scripting. I'll briefly show how it was done.

Finally appendix contains useful links and screenshots.

## 2. Dashboard Light

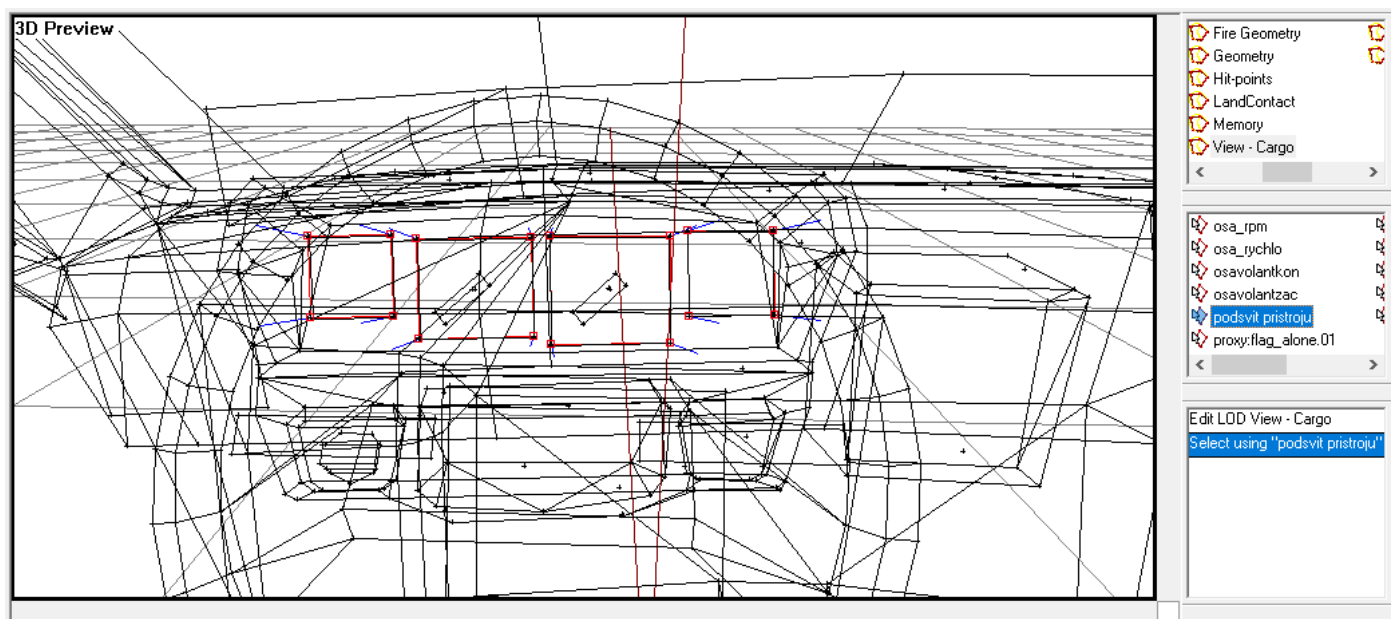
During nighttime the dashboard lights up. Let's take one of the cars as an example:

Sports Car - class Rapid - data3d\rapid.p3d

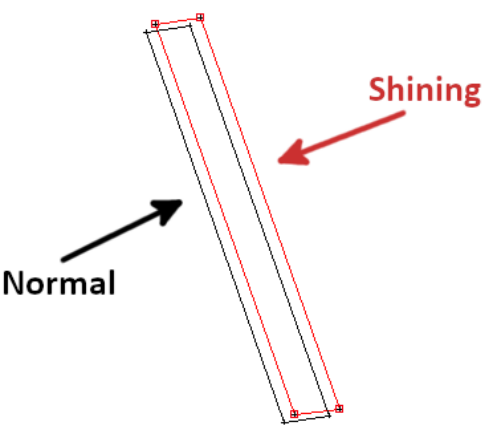


The model has two faces for dials: one normal and one with enabled lighting. The game hides the latter during the day. When the vehicle engine is on and it gets sufficiently dark (which depends on the day of the year; for May 10<sup>th</sup> it's 18:11-05:30) then it reveals the lit section.

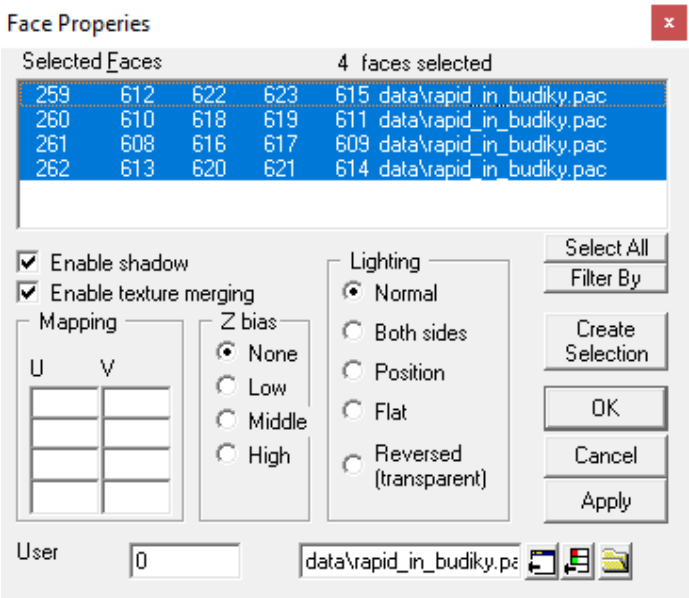
The game looks for a selection called "podsvit pristroju" (which means "backlight" in Czech language). Here's the model in Oxygen (Objektiv2 Light) with this selection marked in red:



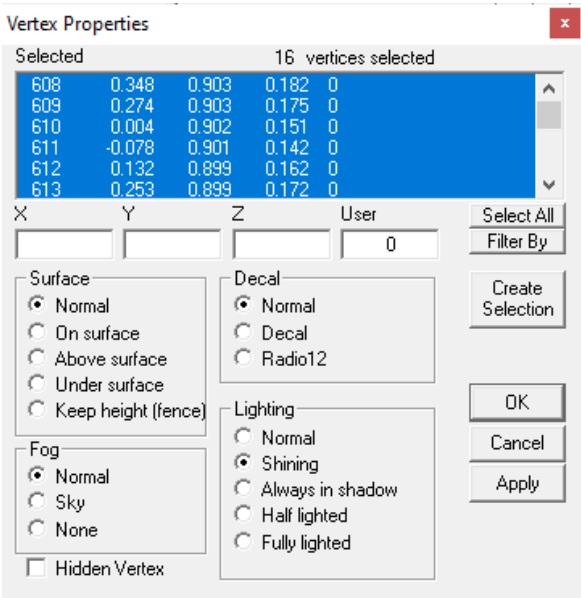
Face duplication can be seen from the side view:



Below are face properties and the texture used:



And here are point properties where the lighting is set:



### 3. Damage Alarm

When a vehicle gets hit hard enough then the game turns on the alarm for four and a half seconds:

- Alarm light blinks five times in 0.5s intervals
- Alarm sound is played

Among original vehicles only AH1-Cobra, OH-58 Kiowa and M113 have the light.

M113 - class m113 - data3d\m113.p3d



It's made identically to the dashboard light except the selection name is "poskozeni" ("damage").

Alarm sound is defined in the configuration the following way:

```
class CfgVehicles {  
    class Tank: LandVehicle {  
        soundDammage[] = {"Objects\alarm_loop1",0.01,1};  
    }  
    class APC: Tank {};  
    class M113: APC {};  
}
```

## 4. Gauges

Gauges in OFP are rotating parts of the 3D model. The game automatically animates them based on the current status of the vehicle (e.g. speed for speedometer). Engine is hardcoded to support only certain types of gauges and it's limited further by the vehicle type (which is determined by the "simulation" property in the configuration).

There are 10 different gauges:

| Name:              | What does it show: |
|--------------------|--------------------|
| horizont           | Artificial horizon |
| IndicatorAltBaro   | Relative altitude  |
| IndicatorAltRadar  | Absolute altitude  |
| IndicatorCompass   | Direction          |
| IndicatorRadar     | Constant rotation  |
| IndicatorRPM       | Engine rotations   |
| IndicatorSpeed     | Speed              |
| IndicatorTurret    | Turret direction   |
| IndicatorVertSpeed | Vertical speed     |
| IndicatorWatch     | Clock              |

Sorted by vehicle type:

| Simulation:     | Which gauges can be used:  |
|-----------------|--|
| airplane        | horizont<br>IndicatorAltBaro<br>IndicatorAltRadar, IndicatorAltRadar2<br>IndicatorCompass, IndicatorCompass2<br>IndicatorRPM<br>IndicatorSpeed<br>IndicatorVertSpeed, IndicatorVertSpeed2<br>IndicatorWatch, IndicatorWatch2   |
| car, motorcycle | IndicatorRPM<br>IndicatorSpeed, IndicatorSpeed2  |
| helicopter      | horizont, horizont2<br>IndicatorAltBaro, IndicatorAltBaro2<br>IndicatorAltRadar, IndicatorAltRadar2<br>IndicatorCompass, IndicatorCompass2<br>IndicatorRPM, IndicatorRPM2<br>IndicatorSpeed, IndicatorSpeed2<br>IndicatorVertSpeed, IndicatorVertSpeed2<br>IndicatorWatch, IndicatorWatch2 |
| ship            | IndicatorRadar<br>IndicatorSpeed   |
| tank            | IndicatorRadar<br>IndicatorRPM<br>IndicatorSpeed, IndicatorSpeed2<br>IndicatorTurret<br>IndicatorWatch   |

## a) Configuration

Gauges are defined in the vehicle's configuration. Inside the vehicle class you add a sub-class named after the instrument.

Here's simplified Sport Car's config:

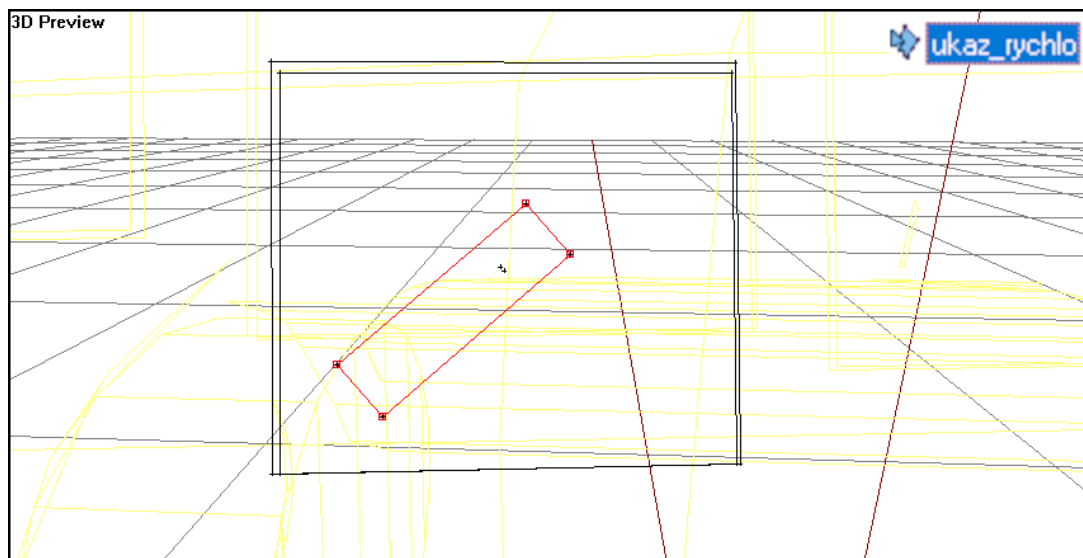
```
class CfgVehicles {
    class Rapid : SkodaBase {
        class IndicatorSpeed {
            selection = "ukaz_rychlo";
            axis      = "osa_rychlo";
            angle     = -260;
            min       = 0;
            max       = 50;
        }
    }
}
```

A gauge class has five properties:

| Property: | Description:            |
|-----------|-------------------------|
| selection | Rotating pointer        |
| axis      | Rotation axis           |
| angle     | Maximum angle           |
| min       | Minimal displayed value |
| max       | Maximal displayed value |

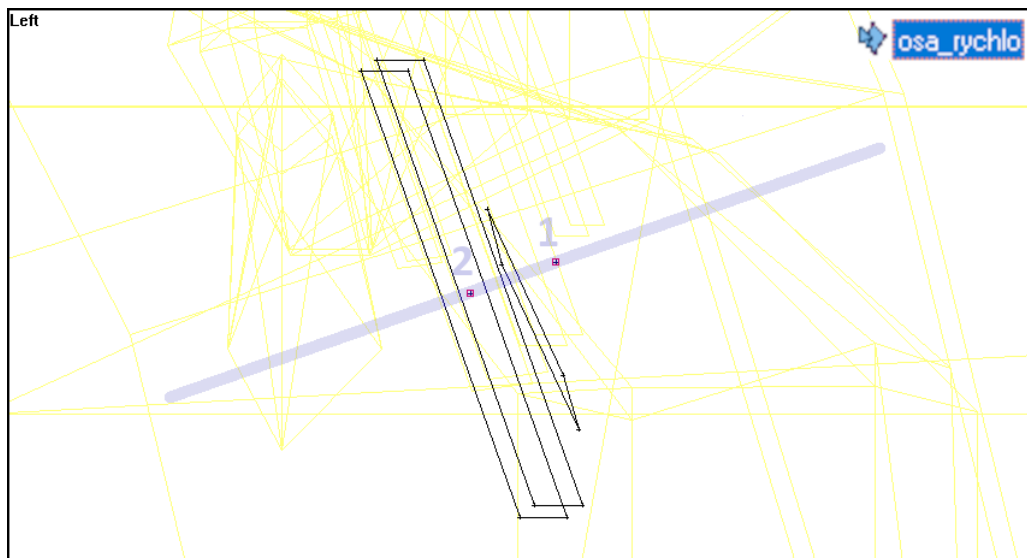
Let's go through them. "selection" and "axis" are names of the selections in the 3D model. They can be renamed.

Below is the speedometer from the Sports Car's model. Selection in red is "ukaz\_rychlo" ("speed indicator") which is used as the gauge's pointer.

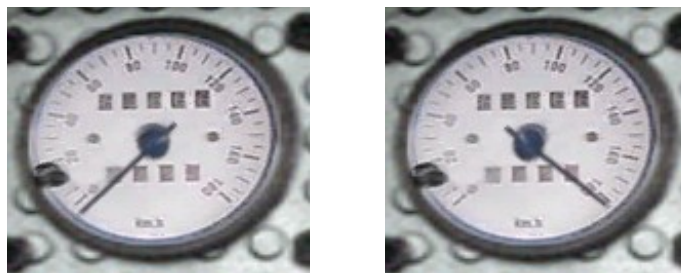


The other selection is "osa\_rychlo" ("osa" means "axis") which consists of two points that make up the rotation axis. The order of points determines turning direction.

I drew a line between the points to make it easier to understand:



Property “angle” determines the maximum pointer turn. Value is in degrees. For the current point order a positive number would make it move counter-clockwise and negative – clockwise. Look at the Sports Car’s speedometer in-game at minimum and at maximum:



The angle between them is 260° degrees (you can use [Wolfram Alpha](https://www.wolframalpha.com/) to visualize any angle):



Finally “min” and “max” determine which speed values the speedometer will display. It’s 0-50 in m/s (0-180 km/h). Pointer at 0° angle (which is set in Oxygen) shows 0 m/s. Pointer at 260° (set by the “angle” property) shows 50 m/s.



## b) Value Range

The range of values that the game uses for gauges is hardcoded and with “min” and “max” properties you can only change what the instrument will show.

Here are value formats listed by gauge type:

| Name:              | What kind of numbers the game feeds it:   |
|--------------------|---|
| IndicatorAltBaro   | Meters  |
| IndicatorAltRadar  | Meters from 0 to 304  |
| IndicatorCompass   | Radians from -3.1415927 to 3.1415927  |
| IndicatorRadar     | Percentage of a circle from 0 to 1  |
| IndicatorRPM       | From 0 to 2 for car, motorcycle, tank. 0 to 1 for airplane. 0 to 9 for helicopter |
| IndicatorSpeed     | Absolute meters per second  |
| IndicatorTurret    | Radians from -3.1415927 to 3.1415927  |
| IndicatorVertSpeed | Meters per second   |

Marked green are classes that can be customized and those in red should probably be left unchanged.

**IndicatorAltBaro** (relative altitude) – shows altitude above the ground level (just like the 2D interface in the top left corner). According to the Commented Config Bohemia Interactive made a mistake with the names. This one should have been called AltRadar instead (and the other one — AltBaro).

**IndicatorAltRadar** (absolute altitude) – shows altitude above the sea level. The game internally divides vehicle’s elevation by 304. Once you ascend above 304 meters it starts from 0 again. That’s why the pointer does not stop (like with most of the gauges) but keeps rotating.

**IndicatorCompass** (vehicle direction) – needs to make a full circle so the “angle” should be kept at 360 (or -360). For “min” and “max” radians are used. 360° is 6.28 in radians but the game shifted the scale into negative numbers.

**IndicatorRadar** (constant rotation) – serves no practical purpose. One revolution takes two seconds. “angle” should be kept at 360 (or -360). None of the original vehicles use it.

**IndicatorRPM** (engine) – the game has its own format for this gauge. The lower the “maxSpeed” of the car the more the engine rotates (up to 2). In the original config “max” is set to 1 which covers the commonly used speed values. Helicopter engine could theoretically go up to 10 but I have not observed it.

**IndicatorSpeed** (speed) – shows velocity in the direction the vehicle is facing. It’s similar to the speed number displayed in the 2D interface except going in reverse counts as a positive number.

**IndicatorTurret** (turret direction) – rotates when a gunner moves the tank turret. None of the original vehicles use it. Properties should be set the same as in the IndicatorCompass.

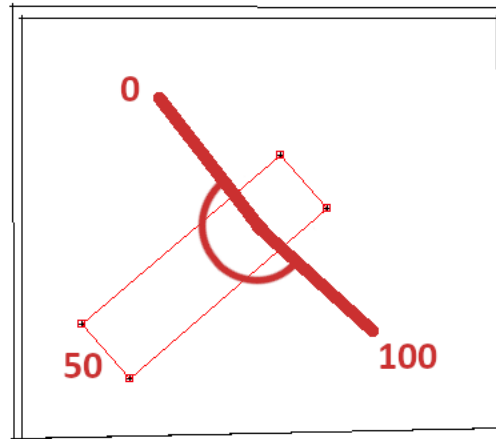
**IndicatorVertSpeed** (vertical speed) – shows velocity in Z-axis. Positive number for going up and negative for going down.



### c) Ships

Gauges for boats work differently because they are hardcoded and the config is not used. IndicatorRadar requires selections "radar" and "osa\_radaru". IndicatorSpeed uses "ukaz\_rychlo" and "osa\_rychlo". Value range goes from 0 to 100 km/h. Angle set in the model actually indicates 50 km/h. Pointer will maximally turn 95° degrees (from the original) in either direction.

If we had transplanted speedometer from the Sports Car into a boat it would work like this:



0 km/h is 95° degrees clockwise from the original and 100 km/h is 95° degrees counter-clockwise.

None of the original boats have any instruments.

### d) Clock

IndicatorWatch is configured differently from all the other gauges. Example:

```
class IndicatorWatch {  
    hour      = "hodinova";  
    minute    = "minutova";  
    axis      = "osa_time";  
    reversed  = 1;  
};
```

Explanation:

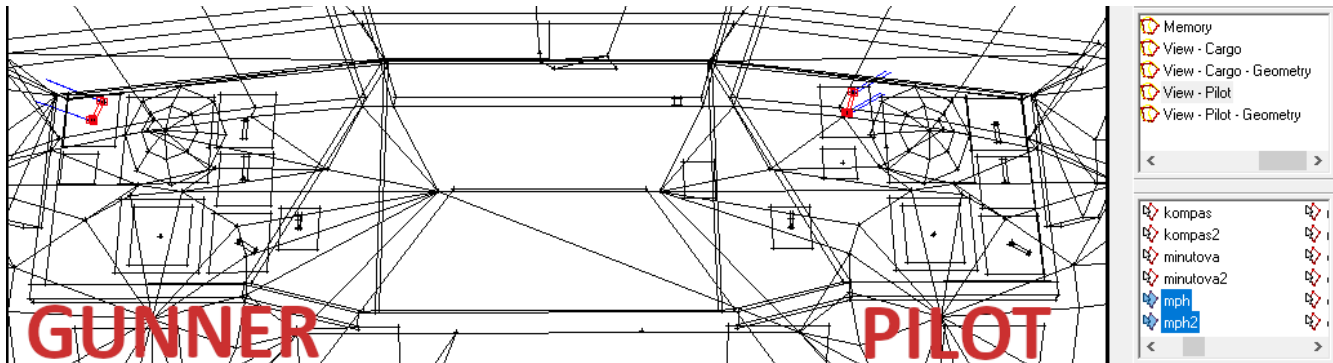
| Property: | Description:                              |
|-----------|---|
| hour      | Selection for the hour hand               |
| minute    | Selection for the minute hand             |
| axis      | Selection for the rotation axis           |
| reversed  | Rotate in the opposite direction (0 or 1) |

Whether the rotation will have to be reversed depends on the point order in the axis selection.

## e) Gunner Seat

Instruments with “2” in their names are duplicates (e.g. IndicatorSpeed2). They function in the same way as the originals (IndicatorSpeed). Their purpose is to be displayed for the other seat. Let’s look at the original Blackhawk:

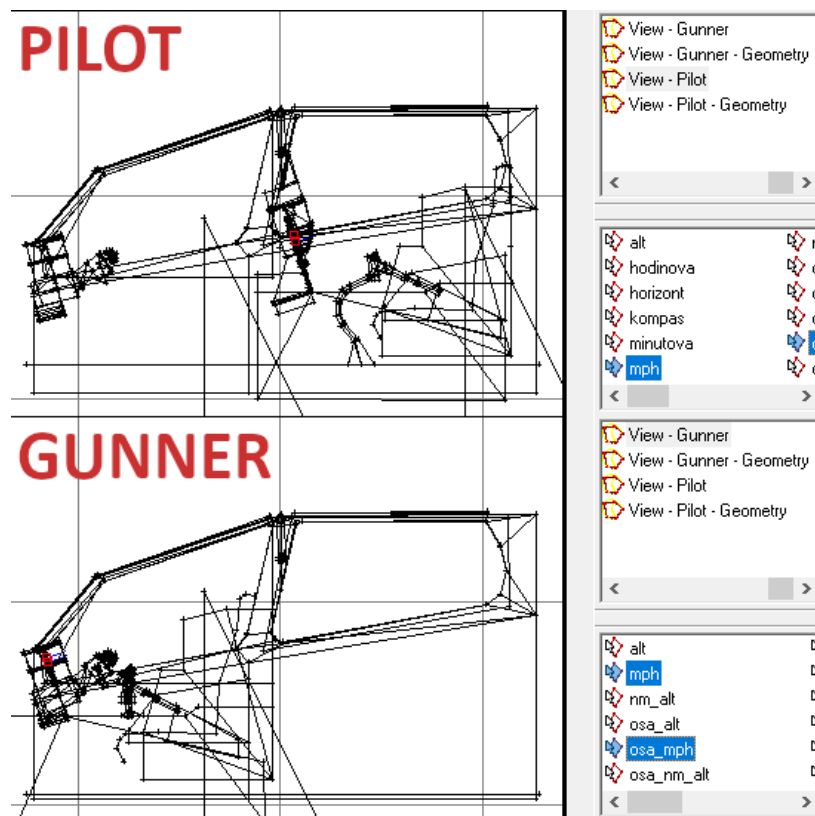
UH-60 - class UH60 - data3d\uh-60.p3d



This is the “View – Pilot” LOD. On the right there’s IndicatorSpeed (mph, osa\_mph) and on the left you have IndicatorSpeed2 (mph2, osa\_mph2).

One LOD cannot have two selections with the same name so that’s why copies with “2” at the end are used. In contrast let’s look at how Cobra was made:

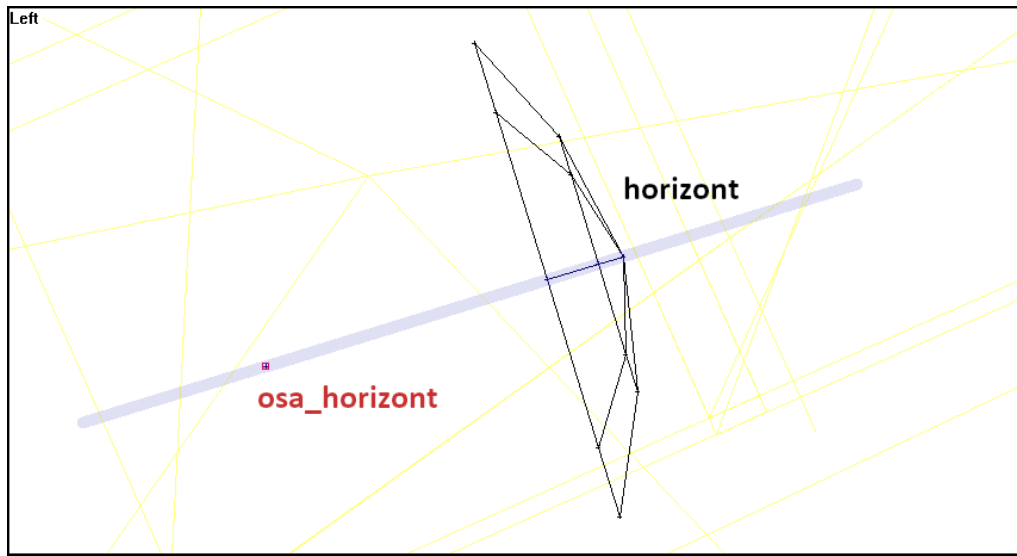
AH1 - class Cobra - data3d\ah1\_cobra.p3d



Pilot and gunner seats have been split into two different LODs so this vehicle can reuse selection names (mph, osa\_mph). They are both handled by one class IndicatorSpeed in the config.

## f) Artificial Horizon

Settings for this instrument are hardcoded and not read from the config. The game uses selections “horizont”, “osa\_horizont” (and “horizont2”, “osa\_horizont2”). Look at the Cobra again:



“horizont” is the rotating textured dome (though it doesn’t have to be a dome) and “osa\_horizont” is used for rotation axis. The latter actually consists of a single point. Axis is a line between this point and the center of the “horizont” selection.

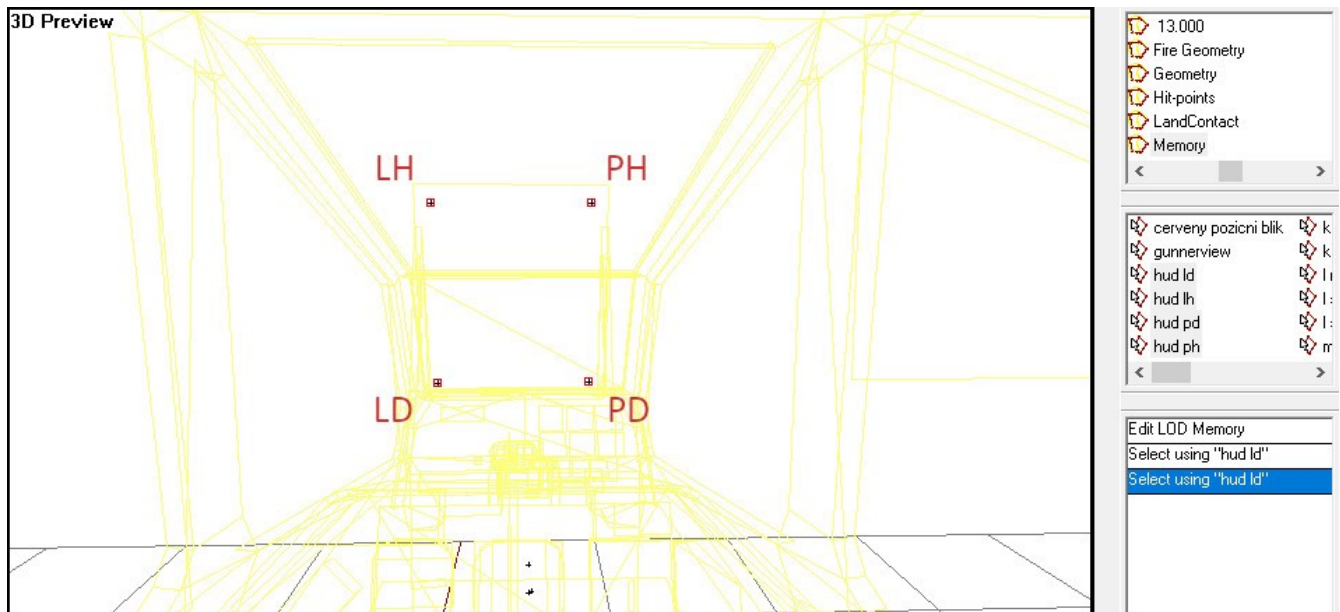
The game rotates “horizont” around the “osa\_horizont” matching the vehicle’s pitch.

## 5. Head-Up Display

Only helicopters can have HUD. The game uses three selections (consisting of single points) from the “Memory” LOD to draw the display:

| Selection: | What does it determine: |
|------------|-------------------------|
| hud lh     | position                |
| hud ph     | width                   |
| hud ld     | height                  |

Here’s how it’s done in the Cobra:



In the background (yellow) there’s “View – Pilot” LOD.

HUD is rendered in half-transparent green (red:0 green:1 blue:0 alpha:0.5). Font “TahomaB48” is used for the digits. Four values are shown:

| Corner:      | What does it show:   |
|--------------|--|
| Top left     | Fuel percentage  |
| Top right    | Distance to the locked target (if there’s a gunner and he’s in your squad) |
| Bottom left  | Speed in km/h  |
| Bottom right | Altitude in m  |



Only two original helicopters have HUD: AH-1 Cobra and AH-64 Apache.

## 6. Custom Instruments

Scripting commands allow you to change textures and rotate parts of the model during the game. This can be used to create custom instruments. I'll examine how [AH64](#) addon by Franze and NodUnit does it. I'll use version 1.2 because it has MLODs.

### a) Changing textures

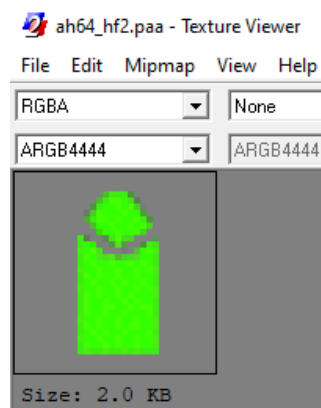
Left Multi-Purpose Display shows how many rockets the helicopter currently has. When you launch Hellfire one of the icons on the display disappears.



This is done by using `setObjectTexture` command. Examples:

```
_ah64d setObjectTexture [16,"\\fz_ah64d\\tex\\ah64_hf2.paa"]  
_ah64d setObjectTexture [16,""]
```

First argument is the selection index. I'll get to it shortly. Second argument is a path to the texture in the addon. If the path is empty then it shows nothing. Here's the relevant image:



Commands are written in the `initialize.sqs` script. It's launched by the `EventHandler` from the config:

```

class CfgVehicles {
    class fz_ah64d: fz_ah64dbase {
        class eventhandlers : ECP_EventHandlers {
            init = "[_this select 0] exec {\fz_ah64d\initialize.sqs}"
        }
    }
}

```

In the helicopter class you'll also find property `hiddenSelections[]` which determines index numbers for the `setObjectTexture` command. It's an array with a lot of items so I'll just show the one we're looking for (item 16<sup>th</sup> counting from zero):

```

class CfgVehicles {
    class fz_ah64d: fz_ah64dbase {
        hiddenSelections[]={... "hudhf1" ...};
    }
}

```

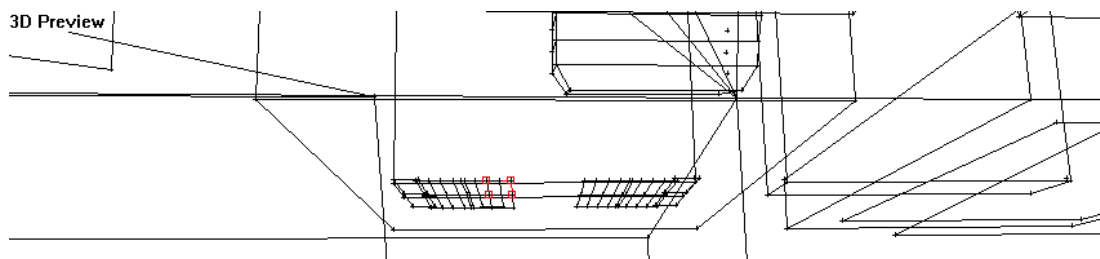
It's also defined in the `cfgModels` in a sub-class named after the P3D file:

```

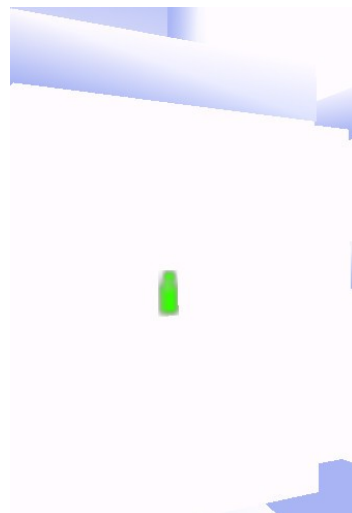
class CfgModels {
    class fz_ah64d_rckt: Helicopter {
        sections[]={... "hudhf1" ...};
    }
}

```

Finally let's look for the selection "hudhf1" in the `fz_ah64d_rckt.p3d` in the "View – Pilot" LOD:



Here's a better view from the external viewer:



To sum up the whole process:

1. Create a face with a texture and then add it to a new selection
2. In the config add the selection name to the `cfgModels` and `cfgVehicles`
3. Write a script that uses the `setObjectTexture` command with that selection's index

At the beginning of the mission these selections are invisible so the command has to be immediately used to show them. Script in this addon runs a loop where the textures are constantly updated based on the amount of rockets left:

```
_ammocnt = (_ah64d ammo "fz_lbhfeight")
? _ammocnt > 7 : _ah64d setObjectTexture [0, "\fz_ah64d\agm114_2.paa"] and _ah64d
setObjectTexture [16, "\fz_ah64d\tex\ah64_hf2.paa"]
```

**Big downside** with replacing textures mid-game is that they might appear blurry. One solution is to preload the image by adding to the model another face with the texture and then moving that face somewhere where it can't be seen.

## b) Rotating selections

Let's examine how the MFD itself works. Being able to seamlessly switch between different screens is impressive.



The action name is "Switch Left MFD" and you'll find in the config:

```
class CfgVehicles {
    class fz_ah64d: fz_ah64dbase {
        class UserActions {
            class switchpltmfdL {
                displayName="Switch Left MFD";
                statement="[this] exec \"'\fz_ah64d\switchmfdpl.sqs'\"";
            }
        }
    }
}
```



In the `switchmfdpl.sqs` script there are commands like the following:

```
_ah64d animate ["lp_mfd",0.5]
_ah64d animationPhase "lp_mfd" == 0
```

The first one runs the animation called “lp\_mfd” and transitions it to a 0.5 state. Number can range from 0 to 1 so in this example it’s half the animation. Second command checks what’s the current animation state. It returns a number from 0 to 1.

Animations are defined in the vehicle’s config:

```
class CfgVehicles {
    class fz_ah64d: fz_ah64dbase {
        animated=1;

        class Animations {
            class lp_mfd {
                type            = "rotation";
                animperiod      = 0.0001;
                selection        = "lp_mfd";
                axis             = "osa_lp_mfd";
                angle0           = 0;
                angle1           = 6.283161;
            }
        }
    }
}
```

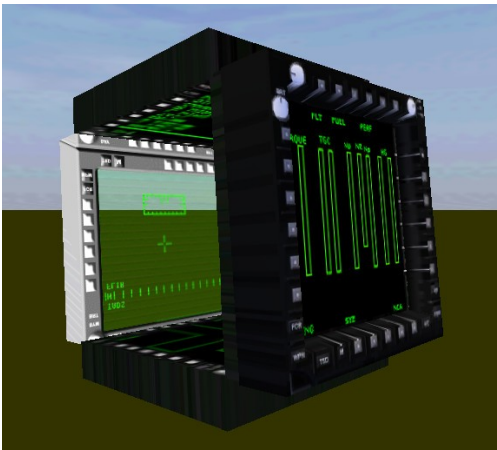
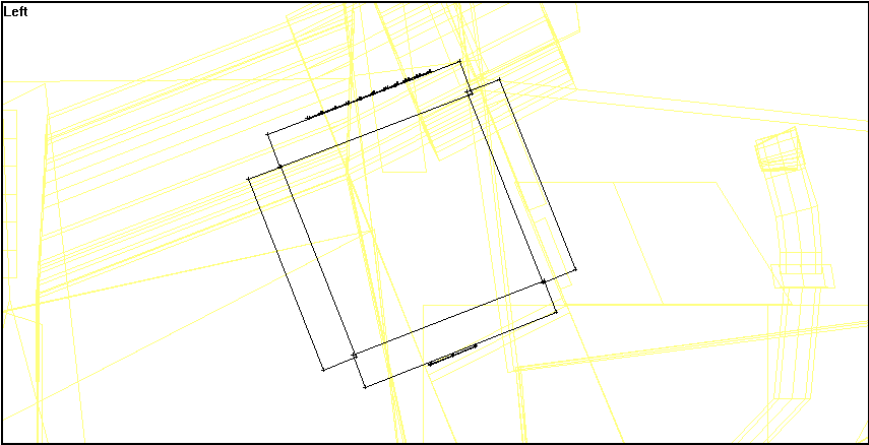
They won’t work without “animated=1” property. Here’s what an animation class features:

| Property:  | Description:  |
|------------|---|
| type       | It has to be “rotation”. Game doesn’t handle anything else. |
| animPeriod | How much time the rotation takes in seconds                 |
| selection  | Selection to be rotated                                     |
| axis       | Rotation axis   |
| angle0     | Initial angle in radians                                    |
| angle1     | Destination angle in radians                                |

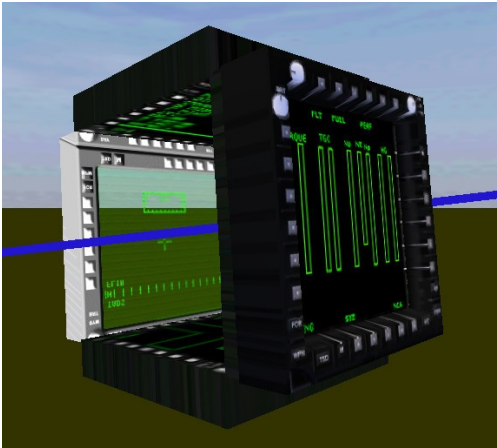
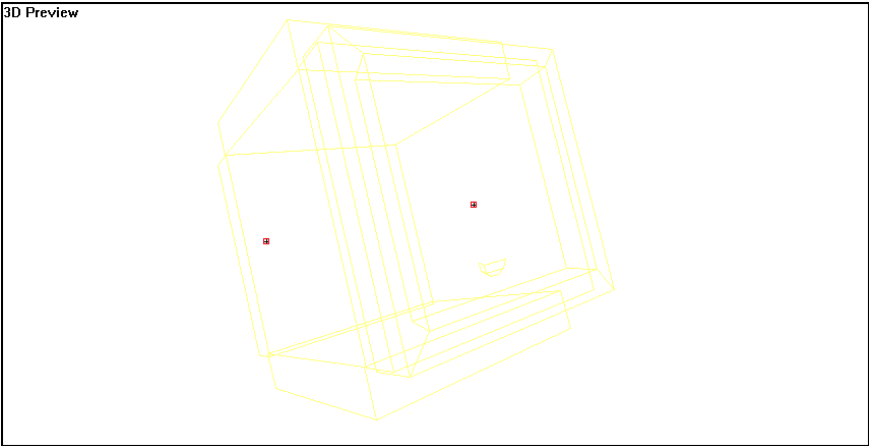
Value for the “animperiod” is very low so the animation is super-fast and that’s why the transition is seamless. “selection” and “axis” you’re already familiar with. Angle goes from 0° to 360° (6.28 in radians) so it makes a full circle. Argument for the `animate` command is a percentage between these two angles. Here’s the numbers chart for this animation:

| animate: | Degrees: | Radians: |
|----------|----------|----------|
| 0        | 0        | 0        |
| 0.25     | 90       | 1.57     |
| 0.5      | 180      | 3.14     |
| 0.75     | 270      | 4.71     |
| 1        | 360      | 6.28     |

Now that we know the selection names we can find them in the model. “lp\_mfd” is in the “View – Pilot” LOD.



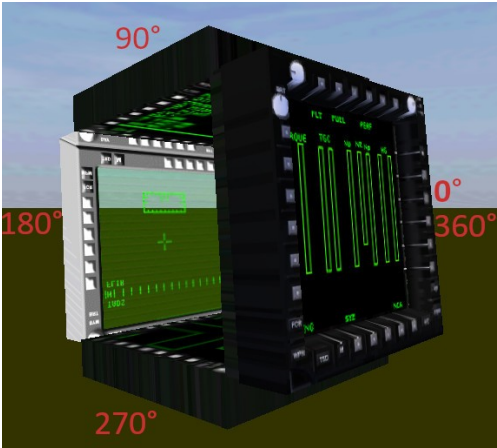
The MFD is simply a rotating cube. Selection “osa\_lp\_mfd” is not in the same LOD but in the “Memory” LOD instead:



Rotation axis goes through the side of the cube like a chicken on a skewer.

The cube has four sides so to shift to the next screen it has to be rotated by 90° degrees ( $360^\circ / 4 = 90^\circ$ ). The table below shows which angles lead to which screen:

| animate: | Degrees: | Radians: | Screen: |
|----------|----------|----------|---------|
| 0        | 0        | 0        | Engine  |
| 0.25     | 90       | 1.57     | Systems |
| 0.5      | 180      | 3.14     | Unused  |
| 0.75     | 270      | 4.71     | Ammo    |
| 1        | 360      | 6.28     | Engine  |



Values 0 and 1 lead to the same result because the cube makes a full circle and lands right back where it started.

In the game the script actually cycles between three screens and not four. One of them is not used. Here's what the code for it looks like:

```
?(_ah64d animationPhase "lp_mfd" == 0.25) : goto "showBLANK"  
  
#showBLANK  
_ah64d animate ["lp_mfd",0.75]
```

If the angle is 90° then it jumps to 270°.

To sum up the whole process:

1. Create a selection that will be rotated
2. Create a selection for the rotation axis in the "Memory" LOD
3. Add animation class to the vehicle's config
4. Write a script that uses the `animate` command with that animation's name

### **c) Is there more?**

I recommend to check other addons made by the same team (including later versions of the AH64). There's numeric speed display, original gauges made to look like a HUD, map showing your position and others.

# APPENDIX A

## Useful links

### Documentation:

[How to Launch Oxygen \(Objektiv2\) Light + External Viewer \(buldozer\)](#)

[Brsseb's model/addon tutorial](#) (see [lesson 9](#) for custom object's animations)

[Official Editing Resources](#) ([link2](#), includes modelling tutorial)

[BIKI – selections translation from Czech to English](#)

[BIKI – animations](#)

[Fab's hidden selections tutorial](#)

[Fab's setobjecttexture tutorial](#)

[Scripting tutorials](#)

setObjectTexture – ([OFPEC](#), [BIKI](#))

animate – ([OFPEC](#), [BIKI](#))

animationPhase – ([OFPEC](#), [BIKI](#))

### Tools:

[Mikero's Extract PBO](#)

[Mikero's Tools older versions](#) (older MakePBO for creating pbos for OFP)

[PBOfilelist](#) (list PBO contents to a text file)

[ODOL Explorer](#) (can be used to preview ODOL files)

[odol2mlod](#) (convert P3D files; list P3D details)

[Official Editing Resources](#) ([link2](#), includes Oxygen and the viewer)

[Lex-OFP tools resource](#) (includes [TexView2](#) for viewing and creating paa files)

[Extracted original configs](#)

[MAS Productions Addons](#) (source for custom instruments)

## APPENDIX B

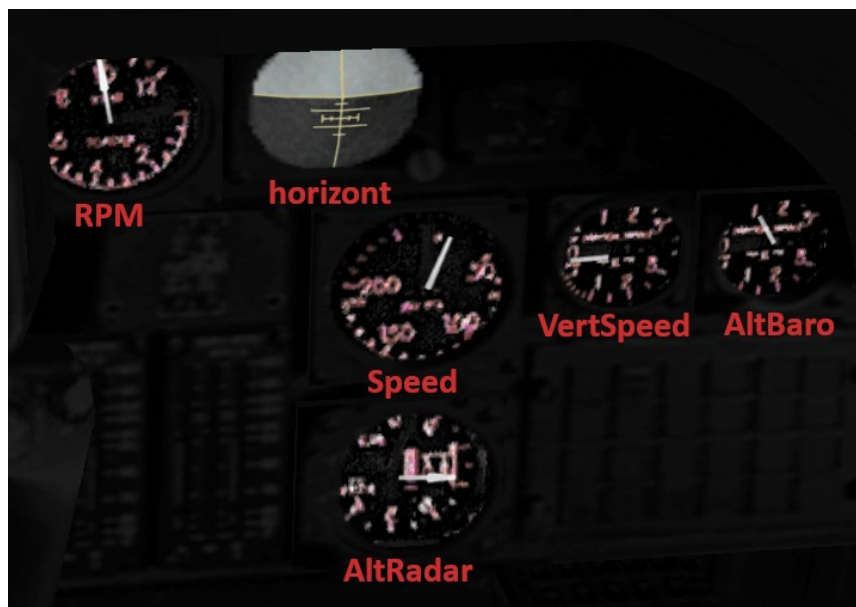
### Instruments in the original planes / helicopters

A10 - class A10 - data3d\A10.p3d



AH-64 - class AH64 - apac\apac.p3d





AH1 - class Cobra - data3d\ah1\_cobra.p3d

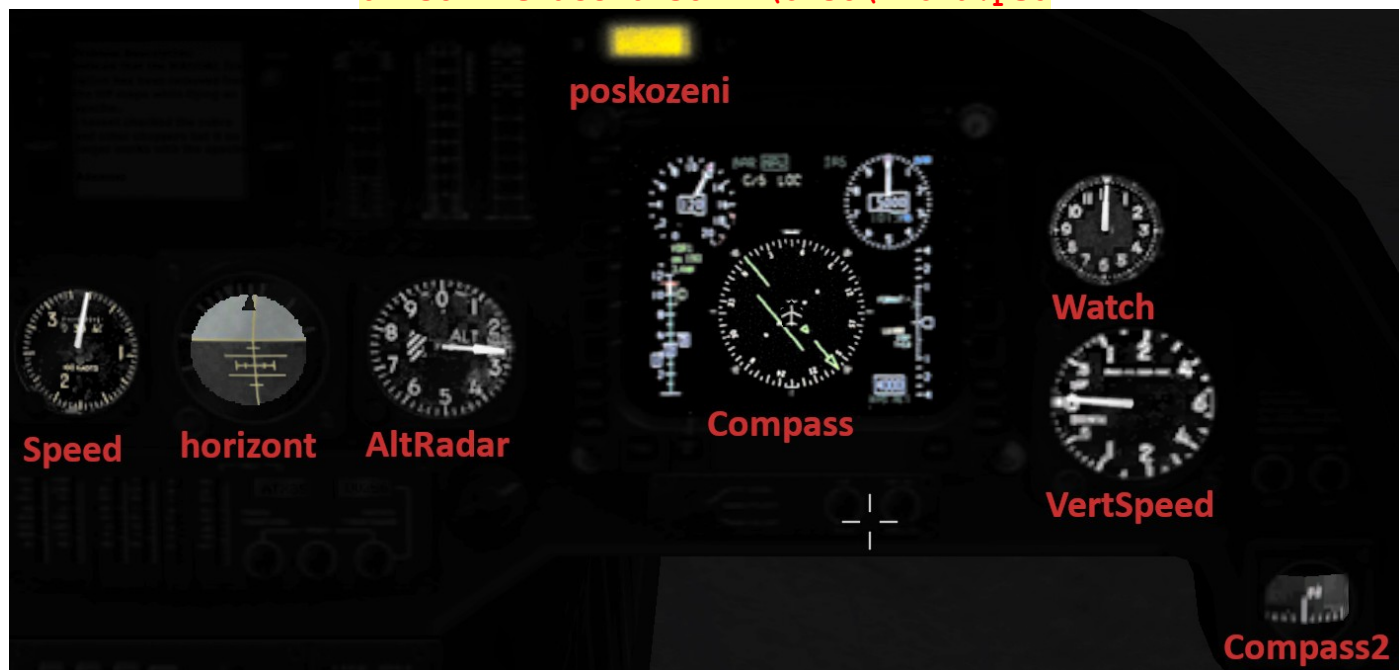


CH-47D - class Ch47D - \ch47\ch-47d.p3d

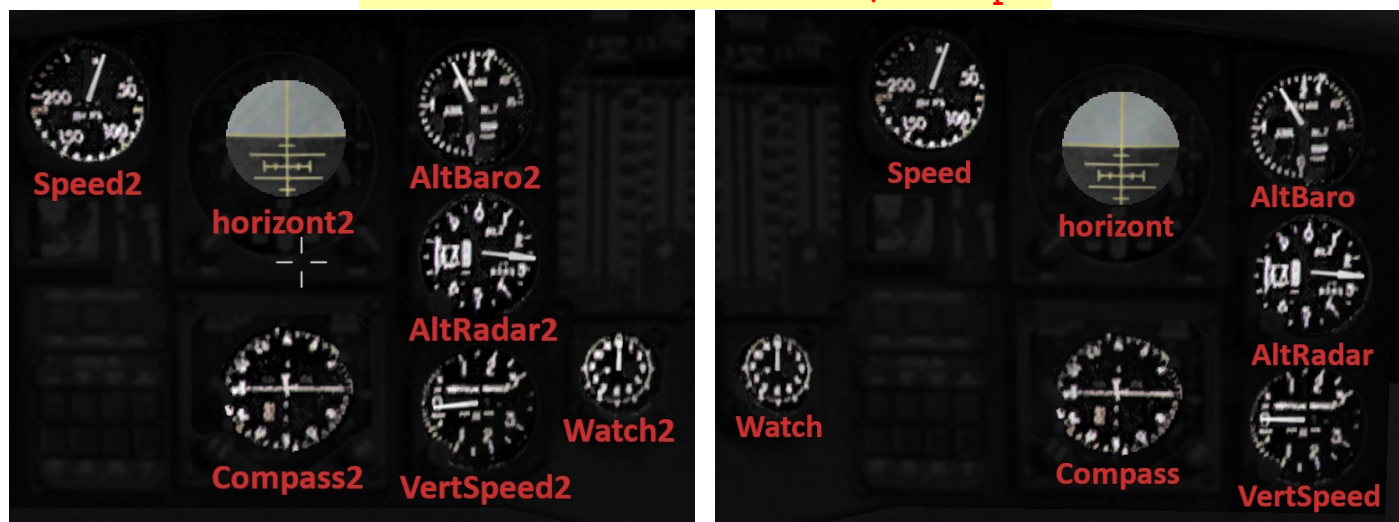




OH-58 - class OH58 - \oh58\kiowa.p3d



UH60 - class UH60 - data3d\uh-60.p3d



Mi17 - class Mi17 - data3d\mi17\_HIP.p3d





Mi24 - class Mi24 - data3d\mi24\_HIND.p3d

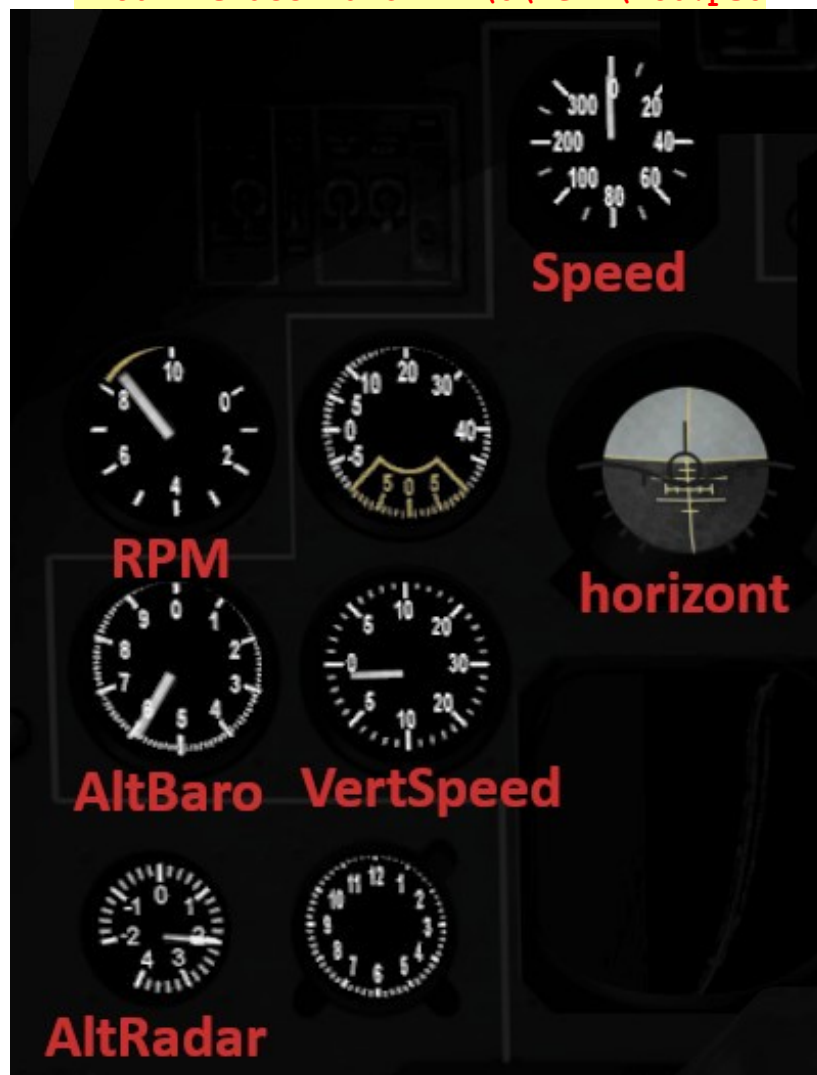




Su 25 - class Su25 - \su25\su25.p3d



V-80 - class Kamov - \O\Vehl\V80.p3d



Plane - class Cessna - data3d\cessna182.p3d



Sopwith F.1 Camel - class BISCamel - \BISCamel\BISCamel.p3d

