DEAD DEER

Principles :
Composed of World and Additionnals (Added) a scene is linked to a pseudo-C script, as main functions OPEN() LOOP() CLOSE().
Clones offer a reuse of Added in building an 2D map environnement or NPC playing, but also Physics or Open Worlds.
Each steps in rendering can be virtualized of made own built. Most of parameters can be modified.

WORLD OBJECTS

nbWO()=3

WO(0) « OBJ »
WO(1) « OBJ »
WO(2) « OBJ »
…

The principle of World Objects is that low-cost precalculated shadows can be applyied,
with even more LIGHTMAPS or DYNAMIC LIGHTING, as usefull to use built-in reflections.
There are 4 modes of basic renderings, Stencil shadowing includes this precalculated shadows.
First shadow mapping use single SHADER linked to, and FULLSHADOWS/SOFT use a
particular segmentation of the rendering pipeline needs. (see examples)

ADDITIONNALS

NbWOADD()=3

ADDED(0) « GROUND »
ADDED(1) « TREE »
ADDED(2) « LEAVES »
…

Check and access to Objects by name, genrally, compose...

```
int getWOByName(char * name)
int getWOByNameNext(char * name)
int getWOAddedByName(char * name)
int getWOAddedByNameNext(char * name)
int checkWOAddedName(int wo,char * name)
int checkMatchWOAddedName(int wo,char * name)
int getWOAddedName(int wo,char * s)
```

More than edition, impostors can be used too until physics, and checked as Morph or EnvMapping.
Note that dynamic shaders must be written conformly to procedure but also to object nature.

Basic renderer can not be sufficient :

int renderSkipDraw(int onoff)	OPEN()	*Do not use basic pipeline.*
int renderDrawScene(int onoff)	LOOP()	*One step in making own rendering.*

Load a Shader :

```
createShader(1,"added_tree.shd");
setNewAssociation(1);
setAssociation(1,"Tex",TEXTURE_NT);
```

TEXTURE_NT ,TEXTURE_NT2,TEXTURE_NL,TEXTURE_NLV,
TEXTURE_BUMP,TEXTURE_0,TEXTURE_1,TEXTURE_2,TEXTURE_3

Shader to World, Added or Material : Exemple : SetAddedShaderContextLightShadowOne(0,1);

```
int resetAddedShaderContext(int n)
int setAddedShaderContextAmbient(int wo,int shd)
int setAddedShaderContextLight(int wo,int shd)
int setAddedShaderContextDaylight(int wo,int shd)
int setAddedShaderContextLightAndAmbient(int wo,int shd)
int setAddedShaderContextLightShadowOne(int wo,int shd)
int setAddedShaderContextLightShadowFourty(int wo,int shd)

int setAddedShaderContextZBuffer(int wo,int shd)
int setAddedShaderContextZBufferMapping(int wo,int shd)
int setAddedShaderContextEdgesDG(int wo,int shd)

int resetWorldShaderContext(int n)
…
int resetMaterialShaderContext(int n)
…
```

Some usefull object context parameters :

```
int setWorldVSParam(int wo,float x,float y,float z,float w)
int setAddedVSParam(int wo,float x,float y,float z,float w)
int setWorldPSParam(int wo,float x,float y,float z,float w)
int setAddedPSParam(int wo,float x,float y,float z,float w)
```

CLONES

As Added can be used for 2D map warpper etc :

```
setCloneWOAsAdditionnal(n,0);                    Simple Added as Clone(n)
setClonePosition(n,pos.x,pos.y,pos.z);
setCloneRotation(n,0,0,0);
setClonePreShader(n,$shaderVoid$);
n++;
```

As usual :

```
int resetClones()
int setCloneWO(int clone,int wo)
int setCloneWOAsMorph(int clone,int wo)
int setCloneWOAsAdditionnal(int clone,int wo)
setCloneWOAddedAsMorph(int clone,int wo)
```

And :

```
void shaderVoid()
{
        if (probe) actualCloneDoNotDraw();
}
```

Basic interactions with keyboard, mouse, VR pads are easy to use :

Mouse :

```
int xMouse()
int yMouse()
int mouseButton(0 or 1)
```

Keyboard and Gamepad :

```
Int Pad()
        KEY_START,KEY_END,KEY_MID,LEFT,RIGHT,UP,DOWN,SPACE,RETURN,
        INSERT,BACK,HOME,DELETE,END,PAGEUP,PAGEDOWN,TAB
int Unpad(key)
```

VR :

```
int toggleDrawVRHands(int draw)

int getVRHandsPosition(int hand,int finger,vector pos)
int getVRHandsInfos(int hand,int finger)
int getVRHandMatrix(int hand,matrix m)
int getVRHandMatrixOrientation(int hand,matrix m)
int getVRHandEulers(int hand,vector v)
```

PSEUDO-C SCRIPT

There are more than 2000 script functions to modelize the whole dream, as functions generally
start with *set*, *get*, *toggle*, *reset* or *verb* and followed by the contextual action.

Scripting parser translate most of basic C style code, and may restricts some purposes.
Basically a use of char* do not implies a full port of pointers, like generally when accessing
variable primitives of function relatives, it uses the name of $VarName$ or « VarName ».

This is used in bufferings, networking, and more.

Moreover, only 3 types of function is allowed :

void func()
int func()
float func()

3 structures is included as global :

vector float3
matrix float16
pixel float2

This implies various forms of methods and use of full rendering options.

Particle effects are named *petals*, there are function for 2D/3D physics, deformables options linked
with bone trees, but also general commands script execution as :

```
GenScriptClean();
GenScriptAdd("SETSEL ADDED");
GenScriptAdd("SELECTED 0");
GenScriptAdd("DUPLICATE");
…
GenScriptAdd("SELECTED 0");
GenScriptDo();
```

This works with main action in the editor, and can be used to generate more scene data as this
can be seen in « broken cube » physics tutorial.