Apostila de Blender 3D

Programando com o Python

Sumário:

Público alvo

Público alvo

Todos os entusiastas ou profissionais que mexem com o programa Blender e a linguagem de programação Python. É necessário ter requisitos básicos de interfaces e funcionalidades do programa **Blender da versão 2.46** e saber programar um pouco em **Python**.

O material aqui contido é uma versão modificada em .pdf dos tutoriais que estão localizados no site http://Blender3Dcarioca.wordpress.com . [Bônus] Exceto o capítulo 7 que é um programa chamado "Script Macro - Cristal Persona" lançado por mim. Ao contrário do Script Macro - AnimPro ver 1.0. Este mesh poderá ser acessado diretamente no ADD\MESH\Cristal Persona.

Basta coloca-lo no caminho Arquivos de programas\Blender Foundation\Blender\.Blender\Scripts\CristalPersona.py

Script (por Rafael Junqueira)
O Script Macro - AnimPro ver 1.0 é um script que renderiza uma animação e gera um TumbNail de 114x57 na pasta Temp\TMP.

Introdução

Este artigo tem por finalidade reunir 4 tutoriais publicados no site Blender3Dcarioca para um formato PDF com um capítulo bônus onde descrevo a criação de uma interface gráfica para interação de um cubo personalizado utilizando um recurso chamado "PupBlock" o script se chama "Cristal Persona" que significa Cristal personalizado.

No **capítulo 1** falo sobre algumas características do Blender.
No **capítulo 2** falo sobre a linguagem Python, e seu objetivo dentro desta apostila.

No **capítulo 3** reproduzo o tutorial de criação de camera disposto no link Programando no Python(Criando uma camera) - 2 (Com alguns acréscimos de informação)
No **capítulo 4** reproduzo o tutorial de criação de objeto disposto no link Programando no Python (Criando uma cena) - 3 (Com alguns acréscimos de informação)
No **capítulo 5** reproduzo o tutorial de criação de fonte de luz disposto no link Programando no Python (Script personalizado - RGB) - 4 (Com alguns acréscimos de informação)
No **capítulo 6** falo sobre a criação de um código que renderiza uma cena com a propriedade de salvar uma imagem ao mesmo tempo. (Recurso do qual o Blender faz separadamente)
No **capítulo 7** é um bônus, e trata de criar uma interface (uma janela) de interação de propriedades de um objeto do tipo Cristal.
No **capítulo 8** diferente da bibliografia listo sites que podem servir de material de estudo tanto para o Blender e Python para acompanhar esta apostila.

Sobre o Blender - Capítulo 1

O programa Blender é uma suíte de funcionalidades que desenham\modelam gráficos 3D, e que podem servir de plataforma para criar animações em diversos formatos de vídeos (favorecendo a compatibilidade com a rede ou com a reprodução em computadores domésticos) e uma enginner (motor) para criar video games (Dando suporte funcional - ou seja apenar mexer em paineis de controles ou procedural, programando na linguagem python) para criar elementos de física e interação (compátivel com o motor Crystal Space).

O programa é de graça e é Open-Source (código aberto, isso significa que usuários podem desenvolver aplicações para serem rodadas no aplicativo Blender. Um exemplo disso são os Scripts que podem ser desenvolvidos na linguagem Python com algum editor de texto e roda-lo juntamente do programa ou disponibilizar o código na pasta de Scripts do diretório Blender.

A sua atual **versão é 2.47**, mas o programa utilizado para esta apostila é a **versão 2.46.** Quasquer testes dos usuários para com os scripts presentes nesta apostila numa versão anterior ou posterior á 2.46 ficará a critério dos mesmos.

O site oficial do programa é http://www.blender.org .

Sobre o Python - Capítulo 2

O python pronuncia-se como "Pai-ton". De acordo com uma palestra ministrada na Universidade Veiga de Almeida, a origem do nome para esta linguagem veio do programa humorístico inglês Python (Monty Python - Em busca do cálice sagrado e entre outros). A linguagem é utilizada pela empresa Google e pela Nasa conforme citados no site http://www.python.org/ e ela é uma linguagem nativa do Blender.

Tornando possível criar funcionalidades ou aperfeiçoar as antigas dentro do programa. Isso significa que o próprio usuário pode batizar uma nova versão dependendo do grau de inovação que o mesmo contribuir para o programa. (Digo batizar no modo particular, para si)

O programa possui uma sintaxe (forma de programar) trivial em relação aos demais programas. Muito embora ficará a cargo de cada um julgar esta facilidade. Ele é um elemento inovativo do Java para o C. Possui as complexidades [comandos poderosos] e a facilidade de programar. Como programador de Java e C, achei o Python muito melhor para

aprender,mas existem obstáculos dentro da linguagem que podem oferecer tarefas tediosas quando surge um erro no script.

A aplicação da linguagem nesta apostila será para computação gráfica.

Não é necessário baixar o python 5.1 ou a versão mais atual para que os scripts citados nos capítulos 3 ao 6 funcionem. Porém para o capítulo 7 sugiro a instalação pois para este não testei-o antes de instalar o pacote de Python.

Para criar um arquivo Python é preciso duas coisas. Uma ter um editor texto qualquer (txt,Word) e um programa que rode como (IDE uma interface de execução, ou um aplicativo compátivel como o Blender). No caso de um arquivo texto, podemos usar o txt\word ou Text Editor disponível no próprio Blender. Abra um novo arquivo nos três casos, e quando salva-lo faça do seguinte modo:

a)Dê um nome e acrescente um ponto e extensão "py" (Ex: Script.py)

Criando uma câmera - Capítulo 3

Todos os códigos foram feitos no Shell do Blender. Shell é uma interface DOS que se perde após fecharmos o Blender. Para que o programa aqui não fique perdido, sugiro que copiem o código e colem no Text Editor do Blender e salvem como arquivo .py.

Como disposto no link Programando no Python(Criando uma camera) - 2 existe um tutorial ilustrativo. Com explicações que o original não possui do código.

Como criar uma camera (Transform Properties,Ortográfica ou perspectiva):

Tópicos de estudos:

a) Como configurar a posição

b) O modo de visão - ortográfico (visão do tipo mapa topográfico) e o modo perspectiva, a visão que geralmente se usa para modelar de modo detalhado. (Ative o teclado numérico com o Num Lock e pressione a tecla 5 para alterar entre Ortográfico e pespectiva)

O Blender permanece a visão no modo Ortográfico como padrão. (**Default**)

Entendendo a classe Camera:

A construtora da classe Camera possui como sintaxe a seguinte forma: Camera.New(Tipo,nome). A explicação da classe construtora. Toda classe conhecida como constructor(construtora) monta um objeto conforme suas configurações.

Quando digitamos Camera.New('persp',"Camera Set 1") estaremos criando uma camera de modo de visão em perspectiva de nome "Camera Set 1". Bem é o ponto importante de toda a questão. Vamos tratar da classe presente neste link CAMERA CLASSE com uma ligeira diferença de tipo, já que no exemplo ele utiliza ortho, utilizaremos Persp, e entre outras coisa que modifiquei para que este tutorial seja inédito e não mais uma repetição.

Definindo a Camera e a construindo: (Camera A)

```
import Blender
from Blender import Camera,Object,Scene #São outras classes que precisaremos

d = Camera.New('persp',"Camera Set 1")
d.scale = 10.0
atual = Scene.getCurrent()
ob = Object.New('Camera')
ob.link(d)
atual.link(ob)
atual.setCurrentCamera(ob)
```

Explicação do código (Camera A) -

Como toda linguagem de programação é necessário chamar as bibliotecas de funções. No caso do C os Headers (**io.h,Math.h e etc.).** O blender deve possuir a chamada de código "From Blender import classe X" porque senão ele não irá puxar as funções dispostas no código. Sem este reconhecimento, o código não é executado, porque ele não existe. Então o primeiro ponto de observação é chamar as bibliotecas que contém classes de funções.

Cada linguagem a chama de um jeito. No caso do exemplo citado, o C é feito da seguinte forma:

#include <io.h> (que inclui o header\cabelaçalho de comandos de entrada e saida)

Ele importa a classe Camera, a classe Object e a classe Scene. (**O que é cada classe está no decorrer desta explicação**)

O comando depois das duas barras **#** chama-se comentário. Não tem nenhum valor em código. Mas serve para orientar o programador ou uma manuntenção por terceiros. O simbolo de comentário modifica pelas linguagens.

Após isso temos um comando que fornece á variavel d (variavel é todo container que pode receber valores diversos) uma camera do tipo de **perspectiva** de nome "**Camera Set 1**". Mas por que atribuir esta configuração da camera á uma variável? A camera não recebe os valores e propriedades diretamente. Ela tem que possuir um "auxiliar". Este auxiliar irá receber os valores no seu lugar, mas ele sofrerá os efeitos das propriedades.

d = Camera.New('persp',"Camera Set 1")

Exemplo: Imagine um conjunto A que engloba o conjunto B. O conjunto B herda todas as propriedades do conjunto A. No caso o auxiliar é o conjunto A. Para os programadores de Flash reconhecerão esta definição como o container.

O próximo caso é a atribuição da propriedade Scale (dimensão do objeto) ou escala que é adotado somente no modo de visão ortográfica. Ela não tem nenhuma utilidade no código acima. Percebemos que **d.Scale = 10.0** é **Camera.new('persp',"Camera Set 1") = 10.0**. Mas como expliquei

anteriormente, a camera não recebe diretamente este tipo de propriedade então ela precisa que uma variavel a receba em seu lugar.

d.scale = 10.0 // Só vale para ortho

A próxima linha é um método que cria a cena 3D. É como se dissesse "Vamos criar um plano 3D para colocar elementos 3D" a variável atual é para armazenar a cena 3D. Poderia ter sido colocada Cena3D ou qualquer outro nome que fizesse menção á função, o que eu recomendo no código é colocar as variáveis com a idéia do método em questão.

atual = Scene.getCurrent()

A variável ob ou objeto ou qualquer coisa que faça menção á um elemento. Vai por fim criar o objeto chamado Camera. Até o d, ela não existia no modo do Scene.GetCurrent(). O objeto é tudo dentro do mundo 3D. Tudo é objeto. Camera,fonte de luz,objeto (Meshes) que é comumente chamados assim. Mas o objeto é todo elemento pertencente ao mundo virtual. Por exemplo, uma pessoa 3D é chamado de Objeto 3D. É uma nomenclatura. Todo objeto recebe um número de propriedades definidas. Neste caso, estamos definindo a camera alocada na variavel d como um objeto. Logo ela passa existir.

ob = Object.New('Camera')

Na próxima linha o ob que definimos que era um objeto do tipo Camera irá "ligar" ou "Conectar" o objeto Criado camera ao d que recebeu a classe Camera. Parece confuso? Imagine que o **d** é um carro. E que o **ob** é a definição que **d** é um carro. **E que o ob admite oficialmente e que receberá todos os atributos de d como um carro**.

ob.link(d)

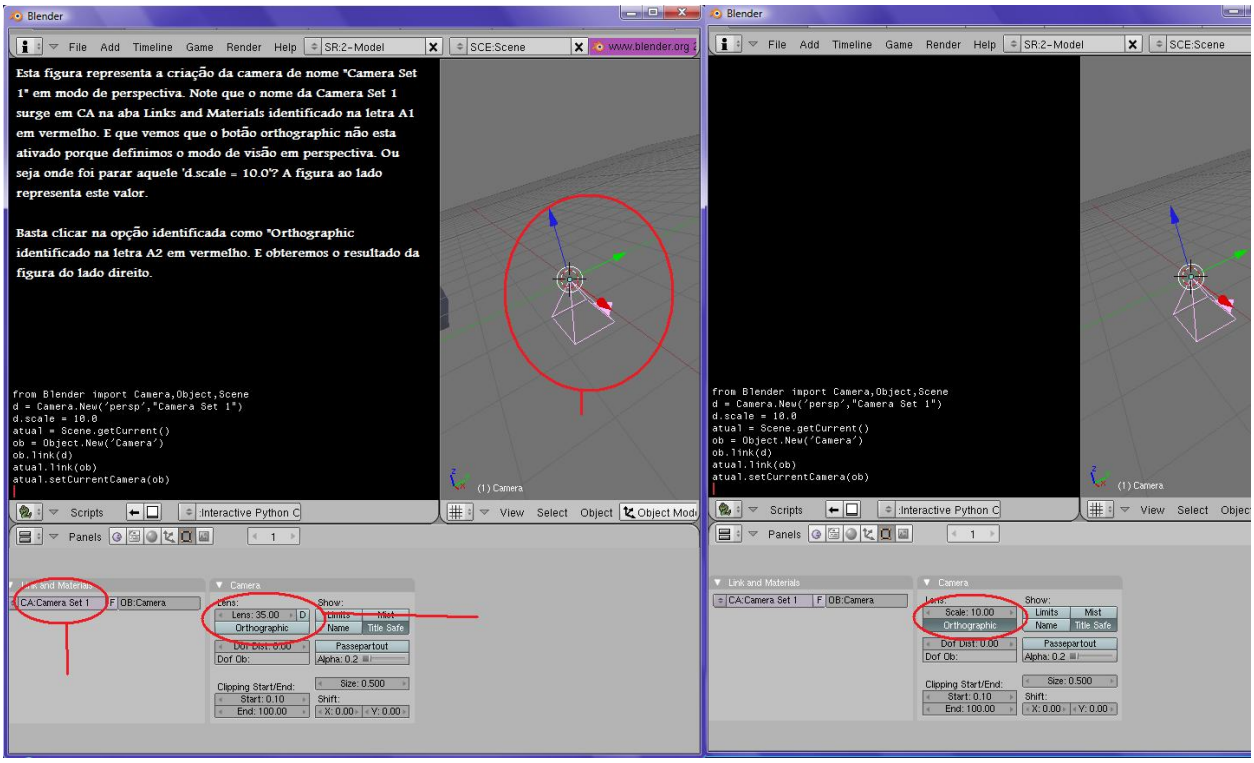
E por vez a cena, que atribuimos para a variável atual irá conectar o ob (Camera sendo uma camera agora) dentro da cena 3D. Ela passará a existir como um objeto 3D virtual. Antes era um conceito. Quando o ob.link(d) ele o tornou uma camera. Quando o atual o conectou ele tornou um elemento existencial.

atual.link(ob)

Na ultima linha temos um comando que determina que a camera ob.link(d) que d recebe Camera. Será a camera da cena.

A camera será criada na posição 0,0,0. E podemos ter uma nova camera. É possivel criar 'n' cameras na cena, serve para aquelas tomadas de cenas complexa que exigem diferentes angulos.

Imagem Ilustrativa da criação da Camera:



Definindo a posição da camera:

```
import Blender
from Blender import Camera,Object,Scene

cena = Blender.Scene.getCurrent()
ob = Blender.Object.New('Camera')
camera = Blender.Camera.New('persp',"Camera Set 3")
ob.link(camera)
cena.link(ob)
ob.setLocation(0.0,0.0,7.0)
Blender.Redraw()
```

Explicação do código (Camera A com posicionamento) -

O que está vermelho sugiro que leia a explicação da **Camera A** anteriormente.

> Existem alguns termos novos neste código em relação ao outro. Só irei citar que os métodos **Blender.Scene.getCurrent(), Blender.Object.New('Camera') e Blender.Camera.New('persp',"Camera Set 3")** é uma versão diferente dos métodos de **Atual = Scene.GetCurrent(), ob = Object.New('Camera') e d = Camera.New('persp',"Camera Set 3")** isso quer dizer que eles possuem o mesmo efeito.
>
> O que vou explicar neste código está localizado nas duas ultimas linhas.

ob.setLocation(x,y,z) define a localização da camera no espaço 3D ou 3D VIEW. Cada coordenada X, Y e Z recebem um valor para localizar o objeto dentro do 3D VIEW.

ob.setLocation(0.0,0.0,7.0) **configura o objeto Camera na posição X e Y (origem) e numa altura de 7.000 (Z).** Se pressionar N no 3D VIEW (com o mouse sobre a tela do Blender) irá surgir uma tela chamada "Transform Properties" que possui três alternadores denominados "LocX,LocY e LocZ" perceba que a camera selecionada apresenta LocX e LocY = 0.000 e LocZ = 7.000.

E a última linha apresenta um comando que substitue o **atual.setCurrentCamera(ob)** que é o **Blender.Redraw()** que significa Redesenho.

Criação de um objeto - Capítulo 4

Como disposto no link Programando no Python (Criando uma cena) - 3 existe um tutorial ilustrativo. Com explicações que o original não possui do código.

Diferente do código acima, não irei criar uma cena inteira (Com camera,objeto e fonte de luz), somente o objeto.
Não colocarei uma ilustração porque a mesma encontra-se com uma imagem não tão nitida.

Inserindo um mesh. (Esfera - UVsphere):

```
import Blender
from Blender import *

#Campo de definição de classes
Esfera = Mesh.Primitives.UVsphere(32,32,5)

#Campo de definição de cena 3D
Cena3D = Scene.getCurrent()

#Campo de definição de objetos 3D
Objeto3D = Object.New('Mesh')
Objeto3D.link(Esfera)
Objeto3D.setLocation(0.0,5.0,0.0)

#Configuração de Cena e Objeto
Cena3D.link(Objeto3D)

#Montagem da cena
Blender.Redraw()
```

Explicação do código (Esfera):

Parte deste código está diferente do tutorial original. Como vimos neste código ele está com comentários que dizem o que cada linha de código representa. Não é recomendável encher

um código de comentários quando já sabemos o que tal comando faz. Este tipo de procedimento é para efeitos didáticos.

As linhas que explicarei será somente a **Esfera = Mesh.Primitives.UVsphere** e **Objeto3D = Object.New('Mesh')** as demais são iguais á camera.

O comando "Esfera = Mesh.Primitives.UVsphere" recebe o valor de uma malha do grupo primitivas denominado Esfera. Mas o que isso quer dizer? Mesh é um objeto geométrico que pode ser manipulado por três elementos básicos: **Edge(aresta)**, **vertex (vertices)** e **faces (face).**

Primitives signifca um conjunto de objetos geométricos pré-definidos. Todo programa gráfico possui um conjunto de primitivas, e a seleção dos objetos segue um padrão comum. Por exemplo no Blender existem as primitivas (**Cubo,Plano,Grid,Torus,Esfera,Circulo,Cilindro e Cone**) e a primitiva especial **Monkey(Suzanne)** no caso do 3D Max Studio (vem as primitivas e compostas) esta última para o Blender vem em forma de Script (**Decaedro,Hexagono**) e no caso do 3D MAX temos **TeaPot** (que é a primitiva composta mais conhecida e comum no mundo 3D).

E UVsphere é uma forma de malha do conjunto primitiva. É um desenho geométrico que relaciona os numero de aneis por segmentos com um raio X. (Em outras palavras: **Rings\Segments\Ratio**) disposto no campo de parâmetros do comando **Mesh.Primitives.UVsphere(rings,segments,ratio).**

Esfera = Mesh.Primitives.UVsphere(32,32,5)

O objeto 3D denominado Objeto3D recebe a classificação de "Mesh" isso significa que ele receberá todas propriedades que uma Mesh possui. É o equivalente do ob.link(d) do código de como criar uma camera. Ou para fins explicativos deste item neste capítulo é transformar o objeto geométrio Esfera em objeto existencial do mundo 3D.

Objeto3D = Object.New('Mesh')

Criando uma fonte de luz - Capítulo 5

Como disposto no link Programando no Python (Script personalizado - RGB) - 4 existe um tutorial ilustrativo. Com explicações que o original não possui do código.

Criar uma fonte de luz é o mesmo procedimento de criação da camera e do objeto. Se você estiver vindo pela primeira vez, e não leu os capítulos 3 e 4, respectivamente criando uma camera e criando um objeto, sugiro lê-los.

Uma fonte de luz é um objeto. Um objeto que deve existir no mundo 3D (elemento existencial) para isso devemos definir uma fonte de luz.O tutorial que segue este capítulo criava três fontes de luzes e três objetos esferas alinhados com cada luz. Como o objetivo aqui é criar uma fonte de luz, vou criar uma do tipo spot de luz azul.

Conceito de propriedades da fonte de luz (Spot\RGB:0,0,1 e Halo=true)

O que está entre parênteses significa o seguinte. Spot é uma propriedade da fonte de luz que gera uma espécie de contorno de cone que abrange uma determinada área. Spot

podemos ter como exemplo, luzes de palcos de teatros ou geradores de penumbra entre as folhas de árvores.

RGB é uma tabela de cor que segue um padrão de combinação para gerar as 7 cores conhecidas e suas 'n' tonalidades. Dependendo do sistema numérico utilizado, o RGB pode assumir as classificações de "Gradiente" ou "Plano simples", a ultima definição não existe. Mas ela significa que teremos uma tonalidade para cada cor. A primeira se chama gradiente, e existe o termo, que define uma tela com diversas tonalidades de uma mesma cor.

Plano simples (RGB = 0,0,1) é azul puro. (Só ele, na mesma tonalidade)
Gradiente (RGB = 150,150,255) é azul com tonalidades claras, tipico de um céu.

Mas o que são estas letras do RGB? Red-vermelho\Green-verde e Blue-azul com estas cores combinadas é possível obter as 7 cores. Dependendo da potência de cada componente do RGB (cada letra é uma componente) teremos uma gradiente. O link acima leva para um tutorial onde três fontes de luzes geram as três cores do RGB, se cruzar uma fonte com a outra, verá surgir outras cores.

Por exemplo se fizer a combinação de RGB sendo 1,1,1 (1.000,1.000 e 1.000) terá cor branca ao contrário cor preta. E assim por diante. Bem este é um básico para curiosidade do funcionamento do RGB e das cores.

Vamos ao código de criação de uma fonte de luz nas configurações já citadas.

Construção da fonte de luz

```
from Blender import *
from Blender.Scene import Render

cena = Scene.getCurrent()

B = Lamp.New('Spot',"Azul")
B.setMode('Square','Halo')
B.setEnergy(2.000)
B.R = 0.0
B.G = 0.0

objetoB = Object.New('Lamp')
objetoB.setLocation(0.0,5.0,10.0)
objetoB.link(B)
cena.link(objetoB)
```

É só configurar a camera para uma altura de 10.0 e posição X = 0.0 e Y = 5.0 e pressionar a tecla F12 para conferir a cena renderizada.

Explicação da fonte de Luz:

O que estiver em vermelho não vou explicar, esta parte está bem mastigada nos capítulos anteriores.

Na primeira linha da cor preta, temos uma variável B que assume o valor da class Lamp com as seguintes propriedades. Spot o tipo de luz e de nome "Azul". Este nome não define a cor ainda da fonte. É somente o nome do objeto.

B = Lamp.New('Spot',"Azul")

A próxima linha define o modo de luz do Spot. O spot é o unico que contém a **propriedade Halo**. **Square** é uma abragência da luz neste formato.

B.setMode('Square','Halo')

Agora é definido a potência da energia de luz. O valor 1.000 é considerado mediano. No caso da fonte de luz do tipo Area, 1.000 é considerado mais forte, por isso os valores 0.250 é o apropriado. Voltando para o Spot foi definido 2.000 (que é 2 não é 2 mil).A potência seria exageradamente imensa. O que ofuscaria toda a visão 3D.

B.setEnergy(2.000)

Esta parte é um entendimento básico. Quando queremos definir uma fonte de luz, é padrão que toda fonte de luz tenha RGB = 1,1,1. Isso significa dizer que, para termos um valor do tipo azul, qual seria nossa configuração do RGB? As componentes R e G com o valor zero? Sim. E vice-versa. É o que observamos no código.

B.R = 0.0 => A lampada de cor azul define que a componente vermelha seja zero
B.G = 0.0 => A lampada de cor azul menos a componente vermelha define que a componente verde seja zero

O resultado é a lampada de cor azul menos a componente vermelha e verde.

Como nos casos da camera e do objeto. Nós definimos que o o objeto terá que assumir o valor da classe. Para que a classe fosse um elemento existencial. Logo teriamos que definir que o objeto era uma fonte de luz.

objetoB = Object.New('Lamp')

Criando uma renderização (e salvando uma imagem) - Capítulo 6

Ao contrário dos demais, este capítulo não possui link para o referido site. Apesar de que no tutorial de RGB e no Script Macro - AnimPro ver 1.0, exista um comando que renderize uma cena e salve uma imagem (como é o caso do TumbNail), preferi refazer o código para que pudesse tomar uma explicação mais detalhada.

Vamos considerar um objeto do tipo Toróide (Torus ou Donut) para fazer parte da cena. E uma fonte de luz, e uma camera. Destes todos, só iremos criar em linha de código o renderizador e salvar uma imagem. Os demais iremos adicionar pelo modo funcional do programa. (Clicando em painéis).

Adicionando os elementos:

Selecione todos os elementos pressionando a letra A, clique em Delete(Del) do teclado sobre as teclas direcionais não numéricas. E diga OK.

Pressione a tecla de espaço e selecione ADD\CAMERA

- Pressione N (Transform Properties) com a camera selecionada
- Defina para LocZ o valor de 7.000 (e deixe o LocX = 0.000 e LocY = -5.000)
- Agora defina para RotX o valor de 90.000 e deixe RotY e RotZ em 0.000

- Pressione a tecla O ou Ins (do teclado numérico) para ter a visão da camera

Pressione a tecla de espaço e selecione com o mouse ADD\MESH\TORUS

- Surgirá uma tela com quatro configurações, clique em Ok sem mudar nada
- Pressione N (Transform Properties) com o Torus selecionado
- Defina para LocZ o valor 7.000 (e deixe para o LocX e LocY o valor de 0.000)
- Defina o RotX para 30.000 (e RotY e RotZ deixe com 0.000)
- Pressione F9 (Editing) e localize na página inferior da tela do Blender uma aba denominada "Link and Materials", clique em Set Smoot (Smooth - suavização)[**Opcional**]

Pressione a tecla de espaço e selecione ADD\LAMP\SUN

- Pressione N (Transform Properties) com a fonte de luz selecionada
- Defina para LocZ o valor de 8.000 (e deixe o LocX = 15.000 e LocY = 0.000)
- Defina para o RotX para 90.000 e RotZ para 90.000 (deixe o RotY em 0.000)

Antes de passar para o código de renderização, vou explicar o que é renderizar. É o processo que calcula os elementos (Objetos de cena, a cena,matrizes,normais) para ser gerado um aspecto virtual apresentável. A cena se resume em linha de código e cálculos matemáticos, mas para efeitos representativos a renderização gera uma cena "apresentável" destes cálculos. (É como a geometria, uma matemática gráfica)

Criando uma renderização

```
from Blender import *
from Blender.Scene import Render

cena3D = Scene.GetCurrent()
IMAG = cena3D.getRenderingContext()
Render.EnableDispWin()

IMAG.render()
```

Explicação do código (Renderização)

Este código não tem ainda um comando para mandar salvar uma imagem. É um código simples de renderização. Se manda-lo executar, a cena que criamos com os elementos de camera,fonte de luz e a torus serão renderizados e mostrados numa tela.

Mesmo esquema, o que estiver em vermelho não tem explicação.

A variável **IMAG** recebe a renderização da cena 3D. Que por sua vez pega a cena existente no comando "**Scene.GetCurrent()".**

```
IMAG = cena3D.getRenderingContext()
```

No comando á seguir temos um formato de renderização. Isso significa que ele vai abrir uma janela que mostra a renderização. Podemos renderizar na tela do programa, ou seja a interface do programa é substituida por um workplace de renderização (Workplace é lugar), ou uma janela acoplada á interface.

```
Render.EnableDispWin()
```

E o último comando gera a renderização.

IMAG.render()

Criando uma renderização + salvando uma imagem:

Repetiremos o código acima, acrescentando mais alguns comandos para salvar uma imagem da cena renderizada. Para explicar este recurso. É só entender que a camera do mundo virtual é como a camera do mundo real. Como ela pode filmar, ela pode tirar fotos. E o tirar fotos é como se revelassemos ela. E podemos salvar uma foto no formato (**JPEG,BMP,PNG,TIFF**), no tamanho (**Resolução 110x110, 1200x1200 e etc.).**

```
from Blender import *
from Blender.Scene import Render

cena3D = Scene.GetCurrent()

IMAG = cena3D.getRenderingContext()

Render.EnableDispWin()
IMAG.extensions = True
IMAG.imageType = Render.PNG
IMAG.sizeX = 640
IMAG.sizeY = 480

IMAG.render()
IMAG.saveRenderedImage("\Imag")
```

Explicação do código (Renderização + Salvando uma Imagem)

No Imag.Extensions é configurado para **True**, porque ele configura que este comando defina os formatos de imagens esteja disponíveis. Estes formatos são o **PNG,TIFF,BMP,JPG**.

IMAG.extensions = True
Na linha seguinte é definido o formato que imagem será salva. No caso é PNG.

IMAG.extensions = True

Na outras duas linhas seguintes, é a resolução da imagem(tamanho). Coloquei um tamanho padronizado de 640 por 480. SizeX entende-se por largura e SizeY por altura. A unidade de contagem é pixel.

IMAG.sizeX = 640
IMAG.sizeY = 480

E a última linha é o comando para salvar uma imagem renderizada. No caso ela vai salvar a imagem na pasta tmp dentro do meu computador. (É possivel modificar esta pasta no Blender). Quando coloca-se **("\Imag")** entende-se que é **("C:\Meu computador\tmp\Imag.png")** , o nome Imag é o nome da imagem que criou-se. Podemos definir qualquer nome.

IMAG.saveRenderedImage("\Imag")

Um fato curioso é que, as configurações definidas no código modificam as configurações dos paineis. Para conferir isso, pressione F10 (Scene) e veja na aba Format a resolução (SizeX e SizeY) e o formato PNG selecionado. Para informação, o formato padrão de imagem é JPG.

Script Macro - Cristal Persona - Capítulo 7

Este é um capítulo de lançamento de Script Macro para versão 2.46 do Blender. Acredito que deva funcionar no 2.47 porque as duas versões parecem ter como ponto de diferenças as correções de bugs que foi notificado pelos desenvolvedores do Blender, e as funcionalidades dos Script pode não ter sido abalada por duas versões que tem poucas modificações de funcionalidades. (As duas foram lançadas no mesmo ano com uma diferença pequena de lançamentos).

Partindo para o contexto do Script, ele se chama "Script Macro - Cristal Persona ver 1.0", e a grande novidade, que são duas é que este Script possui aquela interface que o UVsphere possui quando queremos adicionar uma esfera. Uma caixa de dialogo para definirmos os segmentos e aneis, e o raio. E a outra noticia, é que ao invés de ter que executar o Script pelo Text Editor, disponibilizei para que fosse executado do ADD\MESH\Cristal Persona.

O processo de instalação é simples, vá no local de download no endereço http://br.groups.yahoo.com/group/Blender-Python_B3DC/files/ e leia um pacote .zip com o título "Cristal Persona - Blender3Dcarioca.zip.zip" neste terá um arquivo txt de ajuda (Instalação,compatibilidade). Basta extrair o CristalPersona.py para a pasta **Arquivos de programas\Blender Foundation\Blender\.Blender\Scripts\CristalPersona.py**

Caso esteja com o Blender aberto quando fizer este procedimento, feche-o e abra-o novamente.

Cristal Persona - configurações: - Figura 1

Para os que observarem esta figura tem uma aparência muito semelhante ao símbolo Skedar (do jogo Perfect Dark do console N64). Mas a minha intenção inicial não foi cria-lo, mas sim uma malha que pudesse ser adicionada, e que fosse complexa de certa forma. O código foi relativamente simples para criar esta estrutura. Por isso que a linguagem de programação aplicada ao Blender pode gerar cenários com
elementos complexos até mesmo para quem não tem prática.

Mas ressalto que o código pode ser tedioso, porque encontrei dificuldades em partes infirmas do código que podem não ser encontradas com tanta facilidade. Uma delas consistia em separar o código de linha, bastava alinha-los que ele causava erro de Script. O python parece conter estes tipos de erros. (Sintaxe,posicionamento e lógica).

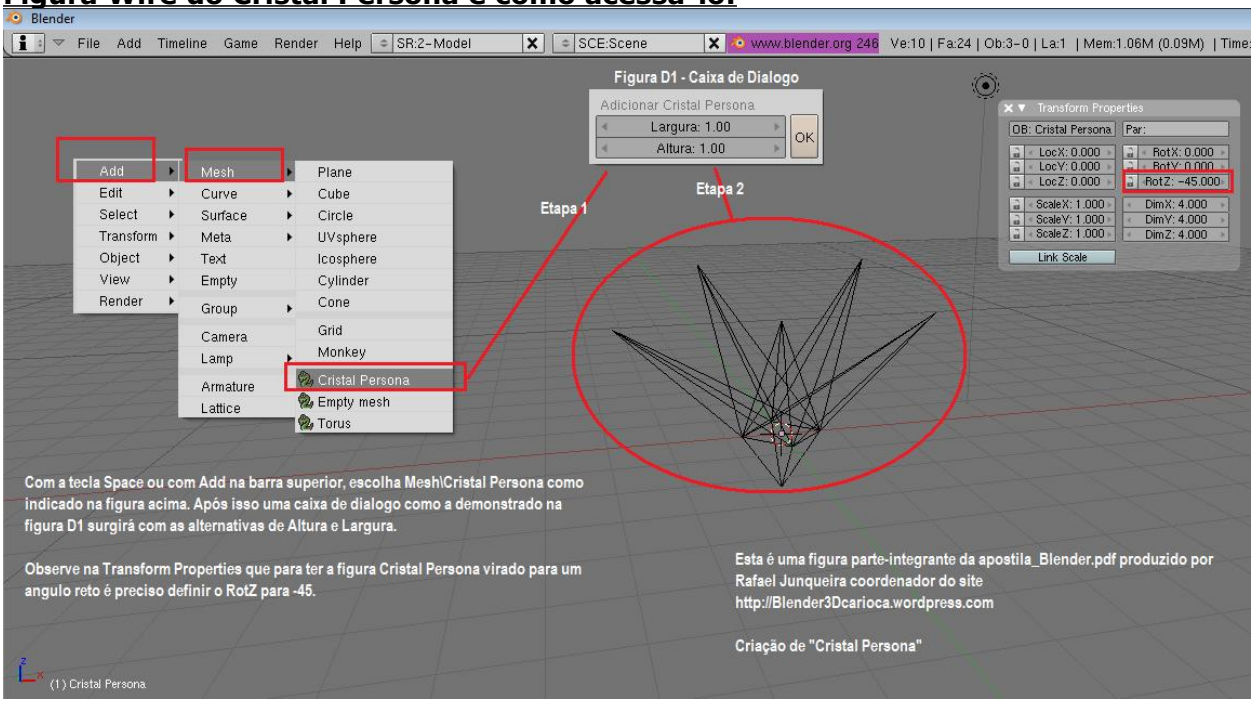
Durante uma palestra soube que um design 3D não precisa saber desenhar para criar os elementos no mundo 3D, mas percebi que para criar um carro é preciso ter uma noção básica de desenho. E de certa forma, até hoje não consegui criar um carro decente. Então se tivesse um Script que gerasse uma carroceria, ou os sistemas e eixos (parte de baixo do carro) não facilitaria? Muito.

Existem muitos scripts que criam primitivas para o Blender, temos o exemplo da Torus (Toróide) que foi um passo para primitivas compostas (tudo que é objeto geométrico complexo, além de uma esfera e cubo) que possua um certo malabarismos com as faces,arestas e vértices.

Cristal Persona de Lado:

As fonte de luz Spot não fazem parte da cena Script. Mas meus futuros projetos preveem criar cenas inteiras com primitivas e cenários inclusive (Como um set,Prop do jogo The Movie) para serem manipulados como se fosse um cenário da vida real. Só com adição personalizada da parte do usuário com elementos, primitivas do Blender e objetos 3D externos.

Figura Wire do Cristal Persona e como acessa-lo:

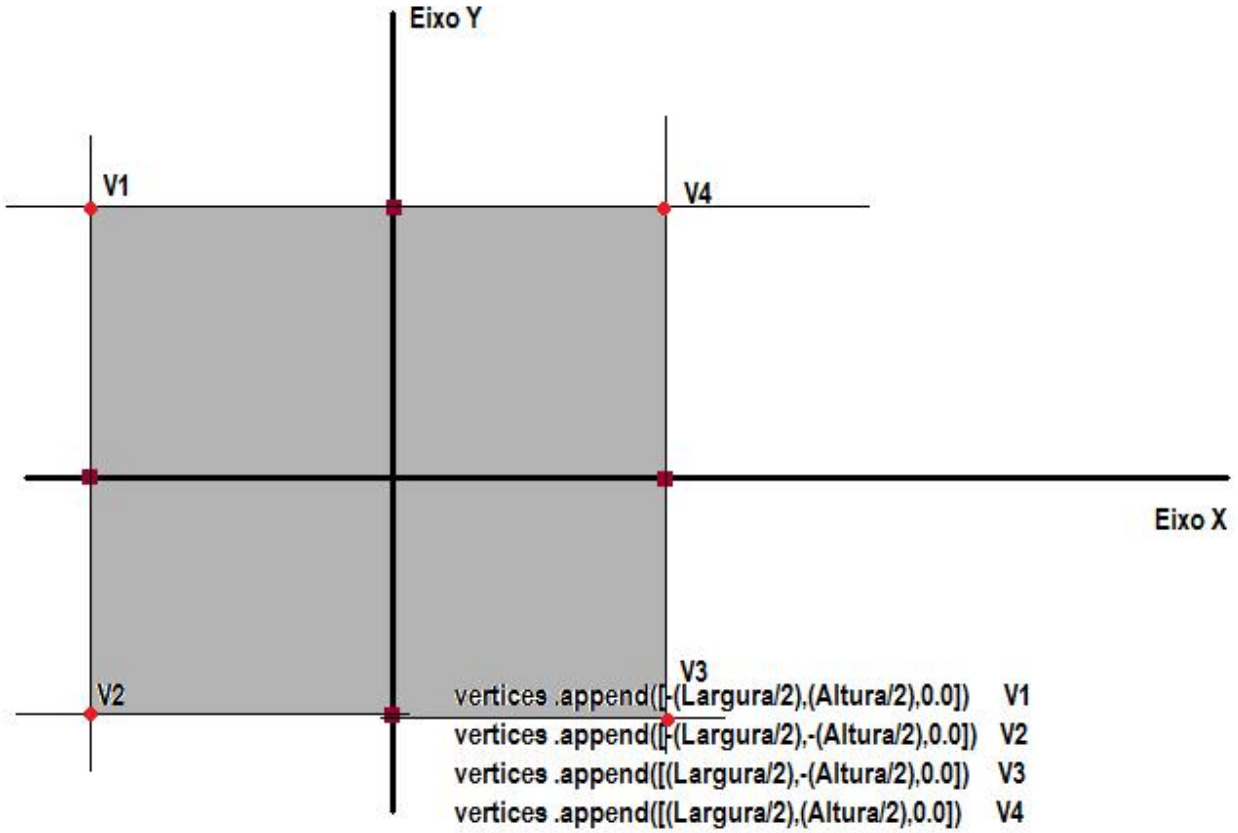


Tutorial como fazer uma primitiva:

A principio o desenvolvedor deve ter em mente o que é vértices e faces, e seus possíveis relacionamentos. Pode parecer incrivel, mas com uma combinação diferente é possivel obter de um cubo á um Teapot. Como fazer um plano? Um plano é desenhado com 4 vértices, 2 formam uma reta. Portanto para formar uma face(plano) é preciso de pelo menos 4. E como fazer isso? Observe o seguinte código:

vertices .append([-(Largura/2),(Altura/2),0.0])
vertices .append([-(Largura/2),-(Altura/2),0.0])
vertices .append([(Largura/2),-(Altura/2),0.0])
vertices .append([(Largura/2),(Altura/2),0.0])

O código acima é um trecho da construção do Cristal Persona. O que ele quer dizer? É um desenho feito no plano cartesiano. Em primeiro lugar a linha de código *vertices .append([-(Largura/2),(Altura/2),0.0])* significa **Anexação de vértices 1 (x,y,z).** Vou representar as coordenadas das quatro linhas acima num plano 2D e verá que se forma um plano (**ADD\MESH\Plane**).



Apartir deste raciocinio é possível montar o Cubo, Torus, Piramide,Escadas,Planetas,Cidades inteiras. O conceito inicial é saber como um elemento 3D se porta no plano cartesiano.
Dica: Mantenha sempre uma folha de rascunho com planos cartesianos para lidar com estes tipos de cálculos. Sabendo disso, é hora de construir as faces. Até o Vertices, se mandar criar uma figura assim só vai aparecer os vértices. Para isso devemos utilizar um comando parecido com "**faces.append([])"**

Antes de partir para este próximo passo, vou escreve um código semi-completo:

```
import BPyAddMesh
import Blender

def PlanoBasico(Largura, Altura):

 vertices = []
 faces1 = []
```

vertices .append([-(Largura/2),(Altura/2),0.0]) -> Equivale ao indice 0
vertices .append([-(Largura/2),-(Altura/2),0.0]) -> Equivale ao indice 1
vertices .append([(Largura/2),-(Altura/2),0.0]) -> Equivale ao indice 2
vertices .append([(Largura/2),(Altura/2),0.0]) -> Equivale ao indice 3

faces1.append([0,1,2,3])

Explicação do código (Parte de criação de primitivas)

A importaçao de classes do tipo BPyAddMesh permite que haja métodos para definir um sistema de coordenadas em primitivas do tipo Mesh. Ela vai proporcionar que a primitiva também seja incluída no grupo AddMesh do Blender como vai ser visto mais adiante.

Existe uma função básica de Python á seguir (**def nomeFuncao(parametro):**) que vai consruir a primitiva. Bem agora vem a parte importante do algoritmo\Script. Os vertices e faces são vetores. Para maior entendimento, são matrizes de posição unica. Que guardam valores em três coordenadas. Para entender ***faces1.append([0,1,2,3])*** é preciso saber que [] é um sinal de vetor. Quando este método realiza este procedimento ela está fazendo a seguinte operação:

Foi desenhado o V1,V2,V3 e V4 no desenho 2D. Agora vou pegar cada um destes vértices desenhados construir com eles uma face.Cada valor dentro do [] no comando *faces1.append([0,1,2,3]) é um valor de indice. Sempre começa do zero.*

Sabendo isso é fácil de deduzir como se faz um cubo. É uma união de faces conjunto de vértices.

return vertices ,faces1 **O comando anterior é o responsável por gerar a primitiva no 3D VIEW, não exatamente á quem constroi, mas a quem possibilita a construção.**

vertices ,faces1 = CristalPersona(Largura.val, Altura.val) **No final do código existe um comando que pega a função inicial (que construir a primitiva) e a ela adiciona os valores do parametro da caixa de dialogo e retorna como primitiva existencial.**

BPyAddMesh.add_mesh_simple('Cristal Persona', vertices , [], faces1) **Este código é o responsável por adicionar o script no menu ADD\MESH.**

- Como construir a caixa de diálogo?

O comando para construir é *Draw.PupBlock(Nome,Opções)* - O código a seguir define um vetor que é o elemento do PupBlock:

block = [] **pode ser dado qualquer nome é um vetor, é um conjunto de elementos quasquer.**

block.append(("Largura: ", **Largura**, 0.01, 100, "Largura do Cristal")) **O vetor anexa á um indice X que largura do vertices.append (anteriormente) receba um valor entre 0.01 e 100.**
block.append(("Altura: ", **Altura**, 0.01, 100, "Altura do Cristal")) **O vetor anexa á um indice X que altura do vertices.append (anteriormente) receba um valor entre 0.01 e 100.**

Observa as palavras em negrito e vermelho. Elas são variáveis. E precisam receber um valor entre 0.01 para que a caixa de dialogo forneça ao objeto Cristal Persona um valor de largura e altura. E como fazer isso. Para os programadores de linguagens (funcionais - tipo Visual Basic) podem ter uma familiaridade neste caso. No caso da referida linguagem bastava que **TextBox1.text = valor**

O caso é parecido, mas usa-se **Largura = Draw.Create(valor X) [no caso do código Blender.Draw.Create(valor X)]**

O código abaixo monta e gera uma caixa de dialogo na 3D VIEW do Blender. Quando *Blender.Draw.PupBlock("Adicionar Cristal Persona",block):* ele está dizendo que uma caixa de dialogo de nome "Adicionar Cristal persona" contendo as opções do vetor Block (Altura\Largura) retornará uma primitiva definida no Cristalpersona.

```
if not  Blender.Draw.PupBlock("Adicionar Cristal Persona",block):
  return

 vertices ,faces1 = CristalPersona(Largura.val, Altura.val)
```

O código abaixo faz jus a funcionalidade do import BPYaddMesh, com o nome "Cristal Persona" ele vai surgir no menu ADD\MESH.

```
BPyAddMesh.add_mesh_simple('Cristal Persona', vertices , [], faces1)
```

Material adicional de Blender - Capítulo 8

Este capítulo é uma substituição da seção bibliografia, que listava alguns documentos referentes ao assunto. Parte do material aqui destacado não foi usado para fazer esta apostila. Exceto o site próprio, e um site que pesquisei sobre PupBlocks, conforme suas seguintes identificações e assuntos.

Ela serve para orientar os usuários novos no Blender.

1. **Blender 3D Carioca - Diretório de tutoriais componentes desta Apostila (Modificados) por Rafael Junqueira**

- http://blender3dcarioca.wordpress.com/2008/03/30/programando-no-blender-python-1/ - Scritps com NMESH (anterior ao 2.46)
- http://blender3dcarioca.wordpress.com/2008/08/05/programando-no-pythoncriando-uma-camera-2/ - Script para criar uma camera
- http://blender3dcarioca.wordpress.com/2008/08/06/programando-no-python-criando-uma-cena-3/ - Script para criar uma cena
- http://blender3dcarioca.wordpress.com/2008/08/07/programando-no-python-script-personalizado-rgb-4/ - Script para criar uma fonte de luz (Composta por objetos)
- http://br.groups.yahoo.com/group/Blender-Python_B3DC/files/ - Download do Script AnimPro e CristalPersona
- http://opendimension.org/blender_en/index.php - Site de estudo sobre criação de primitivas (Inglês)
- http://wiki.blender.org/index.php/Scripts - Download de Scripts no site Wiki.Blender (Inglês)

2. **Diretório de Blender - (Site oficial ou não\nacional ou não)**

- http://www.blender.org/documentation/246PythonDoc/index.html - Referência do código para versão 2.46 (Inglês)
- http://BlenderTotal.wordpress.com - Blog com tutoriais,noticias sobre Blender (Português)
- http://cogitas3d.procedural.com.br/ - Site de Design de Cícero Morais (Aka utilizado Cogitas) (Português)
- http://calebfs.wordpress.com/ - Site de Caleb (Tutoriais de blender) (Português)
- http://www.blender.com.br - Site de Blender (Foruns,contests,estudos,eventos) (Português)

3. **Diretório sobre a linguagem Python**

- http://docs.python.org/index.html - Referência da linguagem Python (Inglês)

- http://www.pythonbrasil.com.br/moin.cgi/ - Referência da linguagem Python (Português)

Dados do autor

Rafael Junqueira.

Sites do autor:
http://Blender3Dcarioca.wordpress.com
http://SahelBlender3D.googlepages.com