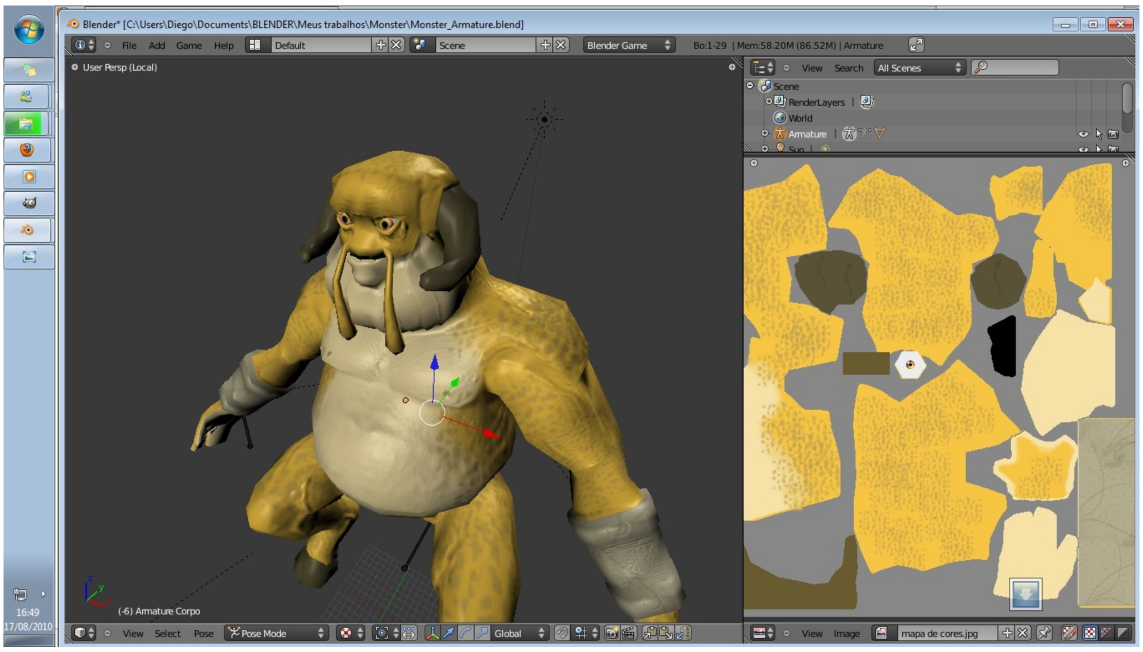
Bending Meshes, Soft bodies, Clothes, Dynamic water surfaces with "Softbody" Blender functionality in real time using BGE (Blender Game Engine)

By "Christian Thomas Oncken" and "Diego Rangel"


Blender* [C:\Users\Diego\Documents\BLENDER\Meus trabalhos\Monster\Monster_Armature.blend]
File Add Game Help
Default
Scene
Blender Game
Bo:1-29 | Mem:58.20M (86.52M) | Armature
User Persp (Local)
View Search All Scenes
Scene
RenderLayers
World
Armature
(-6) Armature Corpo
View Select Pose Pose Mode
Global
View Image mapa de cores.jpg
16:49
17/08/2010

Abstract: The main base of this systems (soft bodies), when applied to the overall game media, has been turning to be an important strand for use in the games of the new generation, so, when using the already existent functionality in "Blender 3D", we decided to make a tool, then we could select all of the Vertex(es) that will be affected by the "Softbody" "Blender 3D" functionality. To achieve this, we used the "Vertex Paint" Blender functionality and a helping Python Script, and when we use "Vertex Paint" functionality in "Blender 3D" in conjunction with the Python Script, all of the already painted Vertex would be linked to "Constraints", turning those Vertexes into "Static", giving the option to use them in a Rig, and thus giving the possibility to "Rig" a "Softbody" Object in "Blender 3D". The tool requires improvements and automation, but for now, that's what we achieved.

Topics :

1) The Python Script works using the “Vertex paint” functionality to make a call for the Vertexes in order to link them with “Constraints” (those Vertexes may or not be linked to an external Object), thus giving the possibility to hook or attach to the Object in a transparent or invisible manner, in order to move the Rendered Object strategically.

This is the self-explained Python Script (commented lines):

```
##23 COLORS USED
## OBJECT COLOR 01: 661E2D
## OBJECT COLOR 03: 54498C
## OBJECT COLOR 05: 764080
## OBJECT COLOR 06: 1C7380
## OBJECT COLOR 07: 979A2A
## OBJECT COLOR 08: 9A4B08
## OBJECT COLOR 09: 9A1812
## OBJECT COLOR 10: 9A565C
## OBJECT COLOR 11: A0CD00
## OBJECT COLOR 12: 35CD00



cor01 = -13820314
cor03 = -7583404
cor05 = -8372106
cor06 = -8357543
cor07 = -13985129
cor08 = -16233574
cor09 = -15591270
cor10 = -10725734
cor11 = -16724576
cor12 = -16724683


# GameLogic Module Importing and important instances
import GameLogic as g
cont =   cont = g.getCurrentController()
obj = cont.owner



### Making a Constraint starting with a
### Coordinate chosen by the user
### Input Values: [x: Float, y: Float, z: Float]
### Input Values: [Nome_de_um_objeto: String]
def constraint(vetPos,objLink):
        # PhysicConstraints Module Importing
        import PhysicsConstraints as pc
        # Taking the Phisical id of the Object
        id = obj.getPhysicsId()
        idLink = g.getCurrentScene().objects[objLink].getPhysicsId()
        # Creating a Constraint
        pc.createConstraint( id, idLink, 1,vetPos[0], vetPos[1], vetPos[2])



### Creating constraints in all of the Vertexes that was
### already painted with Vertex Paint
def criaConstraints(objLink, cor):
        # Searching for the Mesh
        malha = obj.meshes[0]
        # Searching for the Vertex Count
        totalVert = malha.getVertexArrayLength(0)
        # Running Listed Vertexes
        for lista in range( 0, totalVert):
                # Searching Vertexes individually
                vert = malha.getVertex(0,lista) ####### CHANGE FROM 0 TO 1
###############
                if vert.getRGBA() == cor:
                        # If the Vertex is already painted, create a Constraint
                        constraint(vert.getXYZ(), objLink)
                if objLink == "H_AnteBracoD":
                                print(objLink, vert.getRGBA())


if "obj_01" in obj:
        criaConstraints(obj["obj_01"], cor01)

if "obj_02" in obj:
        criaConstraints(obj["obj_02"], cor02)

if "obj_03" in obj:
        criaConstraints(obj["obj_03"], cor03)

if "obj_04" in obj:
        criaConstraints(obj["obj_04"], cor04)

if "obj_05" in obj:
        criaConstraints(obj["obj_05"], cor05)

if "obj_06" in obj:
        criaConstraints(obj["obj_06"], cor06)

if "obj_07" in obj:
        criaConstraints(obj["obj_07"], cor07)

if "obj_08" in obj:
        criaConstraints(obj["obj_08"], cor08)

if "obj_09" in obj:
        criaConstraints(obj["obj_09"], cor09)
```

```
if "obj_10" in obj:
	criaConstraints(obj["obj_10"], cor10)

if "obj_11" in obj:
	criaConstraints(obj["obj_11"], cor11)

if "obj_12" in obj:
	criaConstraints(obj["obj_12"], cor12)
```

You can create more links as you need, all you need to do is to add more colors and more Objects linked to those colors.

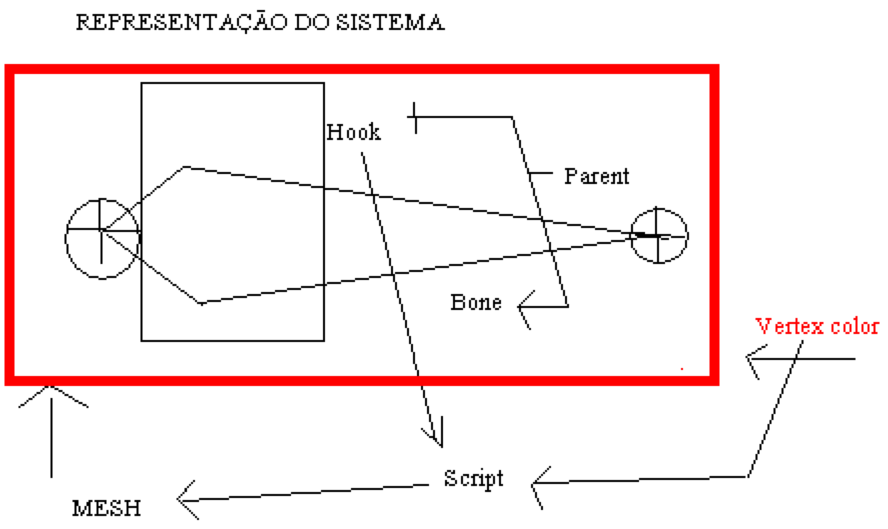
2) There are lots of applications for the given Python Script, you can turn Object parts hardened while others not, and this will make easy for example, operations like pinning Objects whereas other areas remain soft/stretchable. Using the “Vertex Paint” “Blender 3d” functionality you can select strategically what you want avoiding the traditional (and boring) manner, where you have to add and configure constraints one by one. This also gives the possibility to setup colors in a customized way, and you can use “GLSL” (Glide Shading Language), or “Texture mode” without worries about Vertex Painting.

3) The secondary applications have their place in Object Rigging, we can create characters with parts of their body strategically softened (abdomen, bust, thighs), in fact, using an approximation of the same technology found in other game engine physical simulators like “Havok” and “Phys X”; now possible also using “Bullet Physical Simulation Library” even using a non standard approach, and we could simulate dynamic water surfaces, plus plants and trees, stipulating areas whereas a movement should appear.

4) The “Rig” of a “SoftBody” Object in “Blender 3D” have to ordinarily follow some hierarchical setups, those are:

$$\begin{tikzcd}[row sep=small]
\text{Armature} \arrow[d] \\
\text{Bone} \arrow[d] \\
\text{Hook} \arrow[d] \\
\text{Python Script} \arrow[d] \\
\text{Area with “SoftBody”} \arrow[d] \\
\text{Object}
\end{tikzcd}$$

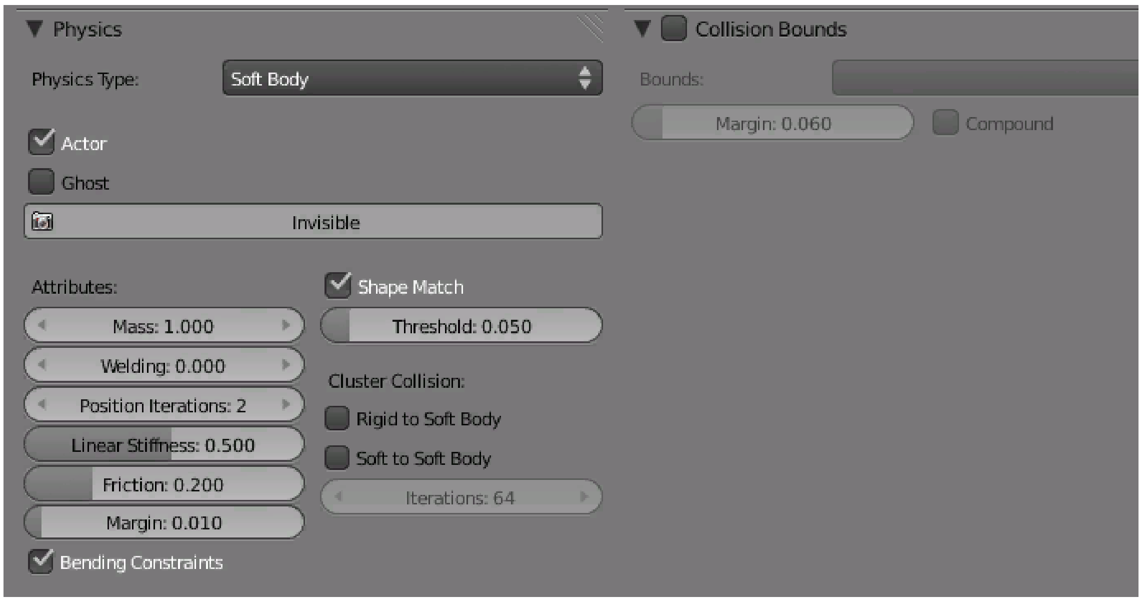
We make a simple draft of the schematic (“Illustrative Image 1”):



Draft of the schematic ("Illustrative Image 1")

Following those instructions, you can create animations and put them on a "SoftBody" Object with the same confidence and quality of a normal "Bone Rig" in "Blender 3D".

5) Performance: This word scares most of the users when talking about Physical Soft Body simulations in games, but with a good setup, they can be used unperceived by the Gamer (from the performance perspective), if you pay attention to those observations:
   a) The Option "Bending Constraints" deserves extreme caution, because it's very CPU hungry, and calculates in all of the Vertexes, this is expensive to old computers.
   b) The "Cluster Collision" Options, for collisions between "SoftBody" and "RigidBody" Objects, have to be used with attention because is expensive in a way that, depending on the Iteration number (lot's of Objects Interacting), could be fatal to the "FPS" (Frames per Second) performance, if you use both of them, in older machines, we recommend to avoid lot's of iterations because those will be softened (the algorithm will auto adjust), at the same time, with CPU expenditure.
   c) The Slider for "Margin" also has to be tweaked to an acceptable value, the more you add to the value, more processing power is required, as the value controls the contact area between Objects.
   d) The Option "Shape Match" may influence some less powerful computers, but it may give an efficient solution for situations when you don't want to use the "Bending Constraints" option.
   e) You have to study a bit more for those other attributes in the tab, but they have less influence in the overall performance of the system as a whole. (See "Illustrative Image 2")


Physics

Physics Type: Soft Body

✓ Actor

☐ Ghost

Invisible

Attributes:
- Mass: 1.000
- Welding: 0.000
- Position Iterations: 2
- Linear Stiffness: 0.500
- Friction: 0.200
- Margin: 0.010

✓ Bending Constraints

✓ Shape Match
- Threshold: 0.050

Cluster Collision:
- ☐ Rigid to Soft Body
- ☐ Soft to Soft Body
- Iterations: 64

☐ Collision Bounds

Bounds:

Margin: 0.060 ☐ Compound

(Screenshot of "Blender 3D" Physics Options) "Illustrative Image 2"

The performance tests, with a base of minimum FPS of 30 (Frames per Second) was done using one machine with less powerful capabilities, a midrange machine, and a workstation/Gaming Machine. We describe those systems bellow:

1. Weak Capabilities:
   - Netbook Acer, Windows XP, Atom CPU, 2,13 Ghz, 3 GB Ram DDR2 , Onboard Graphics (Intel 945 GM)
   - Results: 25 flags , 13 -FPS

2. Mid Range Machine:
   - Notebook Acer, Windows vista , AMD Semprom CPU, 2,40 GHz, 3 GB Ram DDR2, Onboard Graphics (with GLSL support)
   - Results: 60~45 -FPS (monster) , 8~15 (water surface simulation)

3. Workstation/gaming Machine:
   - Sony Vaio,Windows XP ,Intel Centrino dual core 2,0 Ghz, Nvidia Ge Force 7500 Go (with GLSL support) , 1 GB Ram DDR2
   - Results: 60~52 FPS (monster) , 15~22 FPS (water surface simulation)

Bellow, we explain the sequence we used to completely "Rig" a Character:

1. Modeling
2. Texturing
3. Bone and Armature Setup and Positioning.
4. Animate from "rest position"
5. Record initial poses
6. Add "Hooks" (Parenting Objects to the Bones)
7. Name correctly Bones and Hooks for the system
8. Python Script Setup (Giving the Objects the colors of the Script)
9. Paint Vertexes (in "Blender 3D Vertex Paint" mode)
10. Soft Body Setup
11. Add "Logic Bricks" (Always – Python Script)
12. Add all of the "Properties" ("String" Type)
13. Add "Logic Bricks" (Animations, Armature)
14. Give the right Soft Body adjusts (fine tuning)

With this document, are sent some demonstrations of functional systems in “Blender 3D” using “BGE” (Blender Game Engine) using this technique/approach, exemplifying some practical applications using the embedded Python Script.

This project was idealized and executed by:

Christian Thomas Oncken - Project Mentor, system inspector/helping, secondary programmer.

Diego Rangel - Master programming, modeling, texturing.

Thanks to the following persons:

“To everyone that supported this project, including my family, Allan, Karen, Gislene, Gastão, Jessica (my girlfriend), and all of the Brazilian Blender Community, including Ivan (A.K.A. Greylica), Leandro (A.K.A. Cerberus),Junior silva, Jhonny, Vitor Balbio and Letícia. This project is to be shared and widespread, this is all about love, inspiration, care and dedication. A Really “BIG THANK” to my friend and cooperator in this Job Diego Rangel“

Christian Thomas Oncken.

Adapted, Translated and Revised by “Ivan Paulos Tomé” (A.K.A. Greylica).