SNAKE 3D

LORENZO BARTOZZI
STEFANO FIORAVANTI
ALESSIO VERDONE
JACOPO VIVALDI

SUMMARY

Summary 2
Introduction 3
Hierarchical tree 4
Background 5
Ground 5
Lights 6
Snake 7
Apple 9
Constant Movements 10
Snake Movement 11
Pivots' movement 11
Skin's movement 12
Camera 15
Sounds 16
Transition System 17
Sitography 18
Appendix – Objects' list 19

INTRODUCTION

This interactive graphics project aims to implement a reproduction of the famous videogame “snake” in a new 3d version.
In order to implement this new version of the game, we decided to use the tools offered by the library ThreeJS [1]. These instruments allow to build the scene as a big hierarchical tree. In fact, it is rooted with the main “scene” and then all the branches lead to a different object of the world: snake, background, lights, apple and ground.
The main challenge was to reproduce properly a natural and continuous motion for the snake. Different geometrical shapes compose the snake, and we assembled them in order to create a realistic animal which slithers in the grass of the field. The synchronization among all these parts required a huge work which reported also some important problem that we had to solve. The most important of these problem was called “action reactivity”, and it was solved in order to manage correctly the interaction between the grid world, the snake and the user’s commands.
In addition to the snake, we implemented different objects which define and enriches the world where the user plays. A set of light sources was implemented: the most important are a moon and a sun which moves in a semi-naturalistic way and shines the scene. Then we defined three camera views which allow the user to interact with game by playing using different views.
In the end we added a set of music and effects in order to improve the entertaining aspect of the game.

Once the game is ready to be played, a transition system manages the game different interfaces which allow to start the game and report all the settings, according to the user’s commands.

HIERARCHICAL TREE

The *scene* is the main object of the world. A new object can be added at scene as his child. Every object must have a parent (except for the scene) and can have children.

Every object has an own Reference Frame (aka RF), adding an object as a child means connect the RFs of the two objects, it's possible to apply a rotation/translation between the two RFs. Any transformation of an object is spread also to his children.

An object consists of a geometry and a material. The latter is further divided in colour and texture.

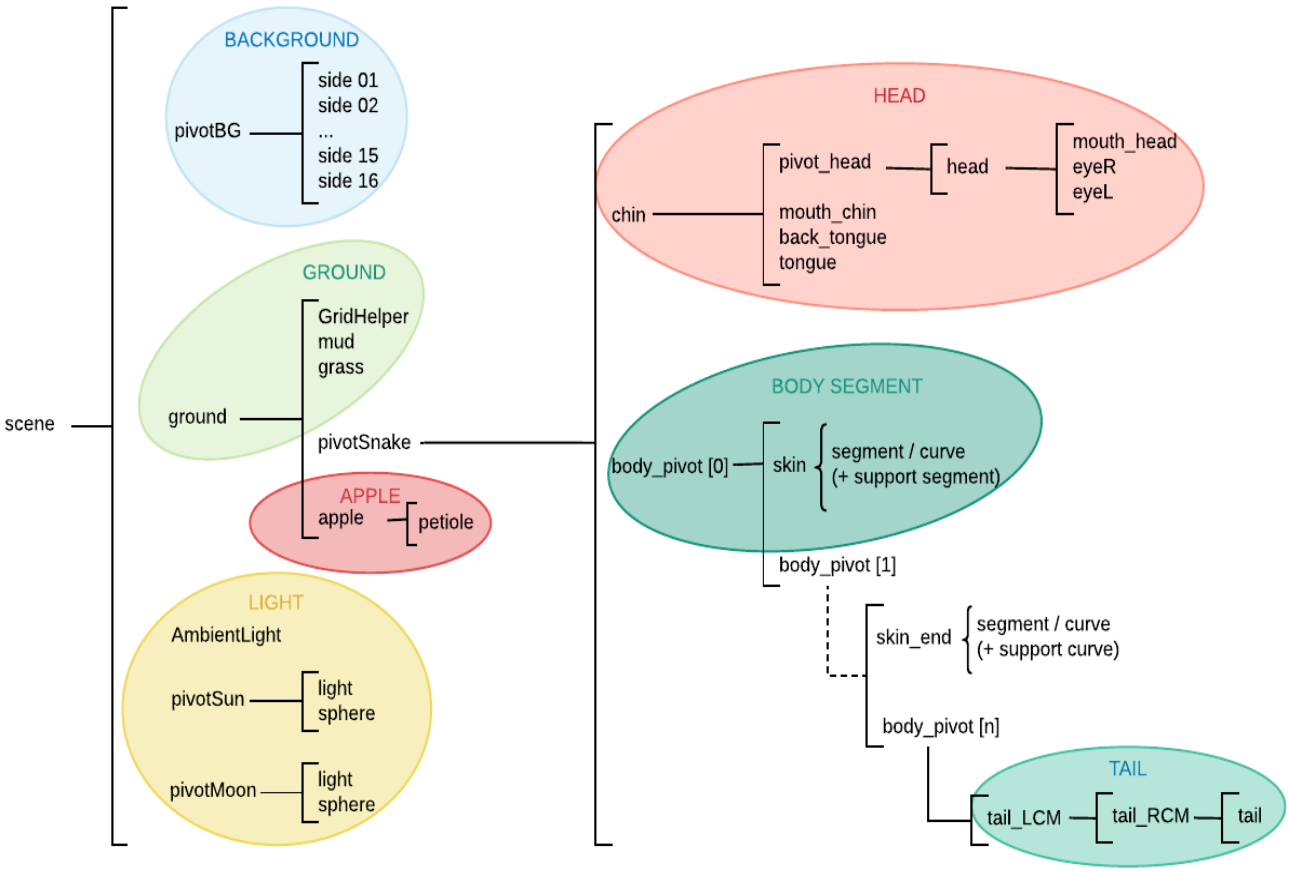


Fig. 1 - Hierarchical tree

BACKGROUND

The background has been realized using 16 concatenated vertical planes. It surrounds all the others objects of the scene and rotates very slowly. A Milky way texture [2] has been applied to each sector; namely, an image of the Milky way has been divided vertically in 4 sub-images and each of them applied to 4 consecutive sectors. This process is repeated 4 times. Note that to obtain a smooth result, the last and the first sub-images are seamless.

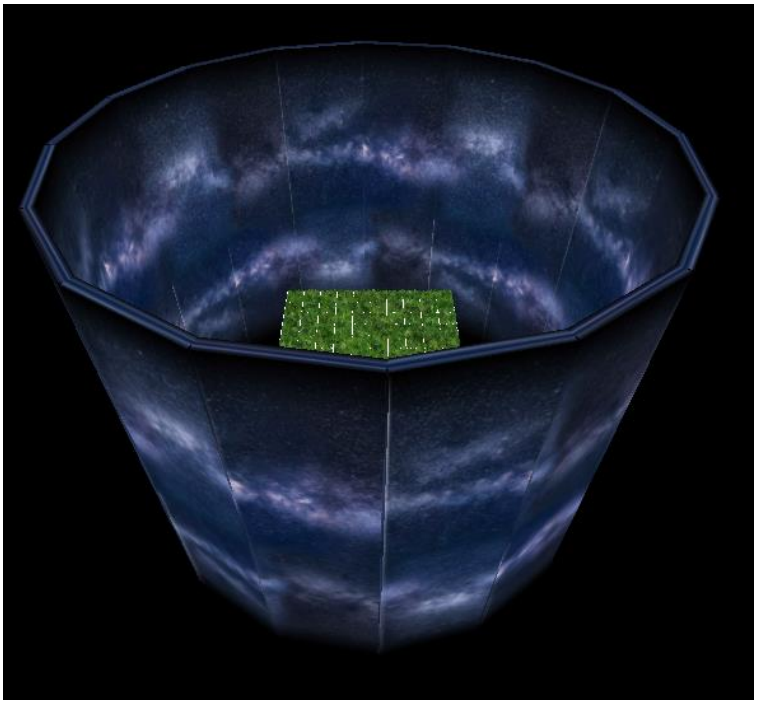


Fig. 2 - Background

GROUND

The *ground* represents the base where the snake can move. It has been built using a box geometry coloured with green; *mud*, *grass* and *grid* objects have been added to it: the latter for the sake of gameplay dynamics, the others to obtain a more realistic effect.

MUD

It has the same geometry of the ground object but a different colour (brown) and has been added under the *ground*.

GRASS

The grass effect has been realized using a plane geometry with 90000 vertices. The z coordinate of each vertex (except for the vertices that lie on the grid) has been set to a random value. This easy method lied to a reasonable result.

GRID

It's a simple GridHelper that divides the plane in a 10 x 10 cells. The apples can spawn only in the center of a cell, and the snake movements are discretized to the cells shape.

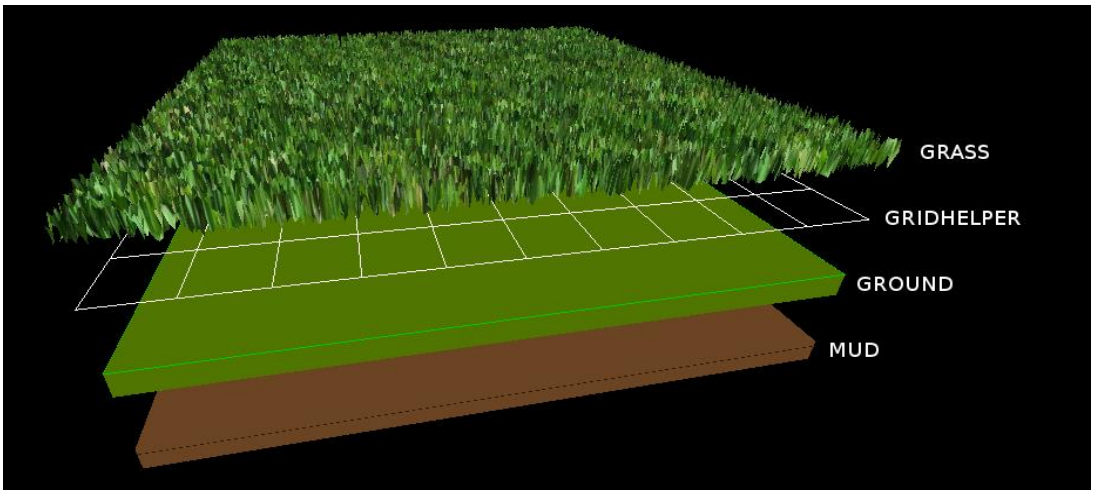


Fig. 3 - Study of ground's components

LIGHTS

In the scene the light's sources are:

SUN & MOON

The main sources of light. In order to obtain a "realistic" effect, they move at different velocities, phases and textures [3]; furthermore, the sun is brighter than the moon.

AMBIENT LIGHT

It's always present, and very soft. It has been added to the scene in order to let the player play also when the main source of light is far away.

SHADOWS

Every object in the scene receives and cast shadows. An exception is made for the internal part of the mouth: casting the shadows used to create unnatural effects, that's why it only receives them.

SNAKE

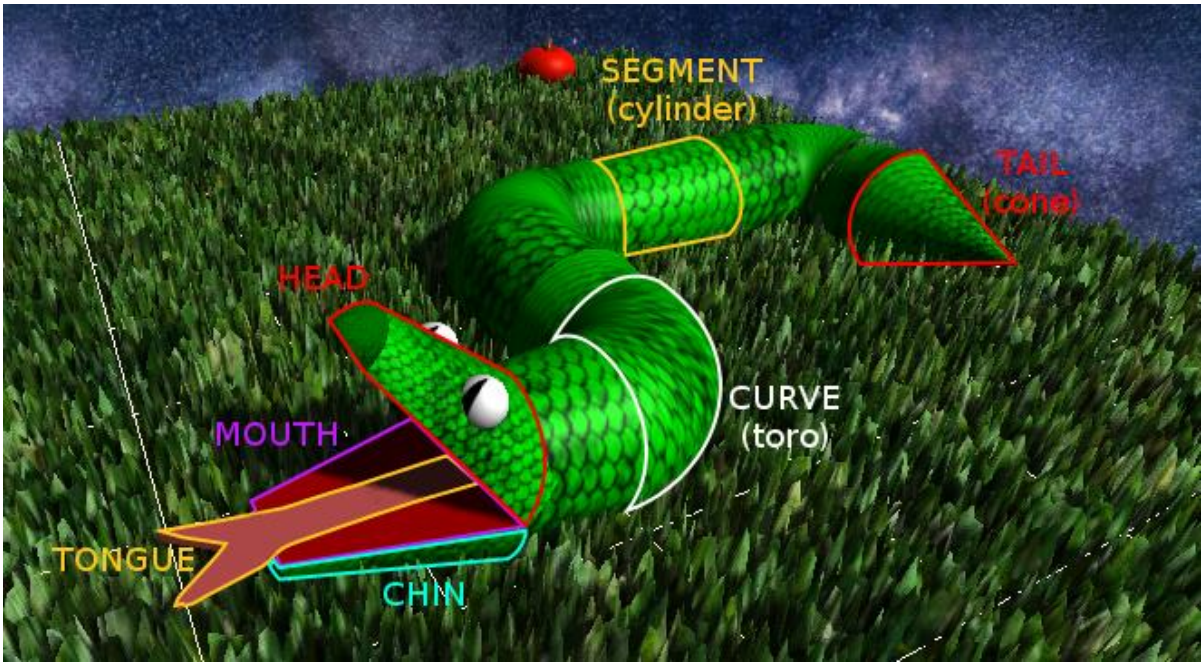


Fig. 4 - Snake with highlighted components

The snake is connected with the ground by the pivotSnake. It can be considered as the root of the snake hierarchical tree, and it's used to move the whole body and to check the game flow.
The snake's body consists of the head and a modular chain of body segment that ends with the tail.

HEAD

The head is the most complex part of scene; it's composed by 8 different objects animated with 2 different motions.

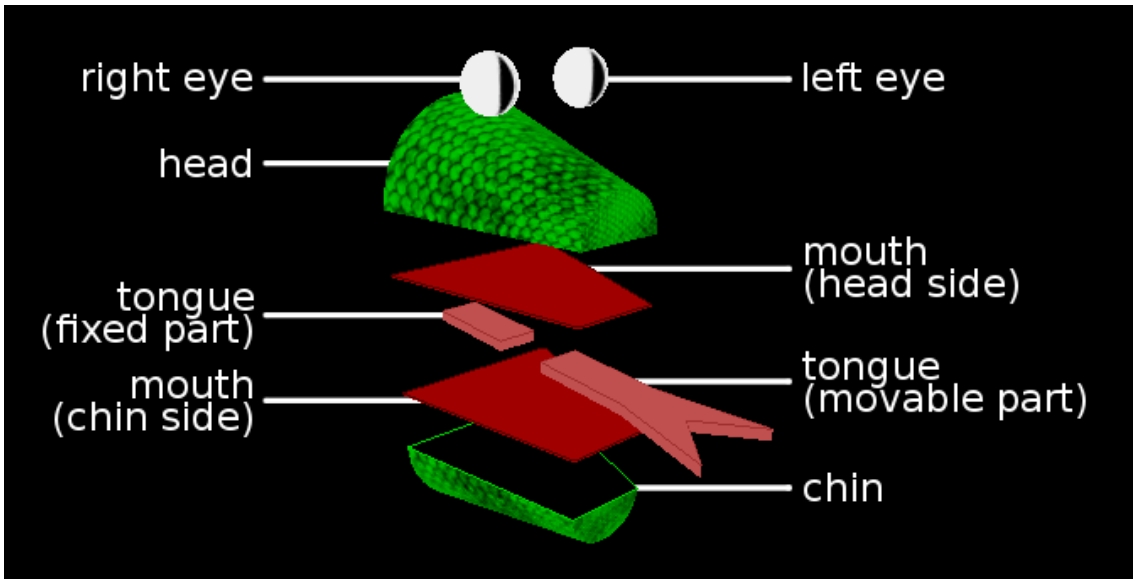


Fig. 5 - Study of head's components

It has been created starting from the **chin**, a cylinder geometry adapted to the graphic purpose.
The **head** is specular to the chin and these two objects are connected by a pivot located at the base of the chin. This pivot allows the open-close motion of the head.
It's possible to see from the overall picture of the head's components (see Fig. 5) that the planar sides of the chin and the head are missing. To face this problem, a new ThreeJS geometry has been drawn, extruded and then added to both the cylinders; these objects constitute the **mouth**.
The **eyes** are represented by two spheres with a texture.
Since the upper side of the mouth and the eyes are added as head's children, they follow the head movement.
The **tongue** is added as chin's child and it must be split in two parts: the back that is fixed and the front that is animated. The split is necessary because
when the head turned, the tongue used to go out of the boundary of the back of the head. Both the tongue's parts are drawn and extruded.

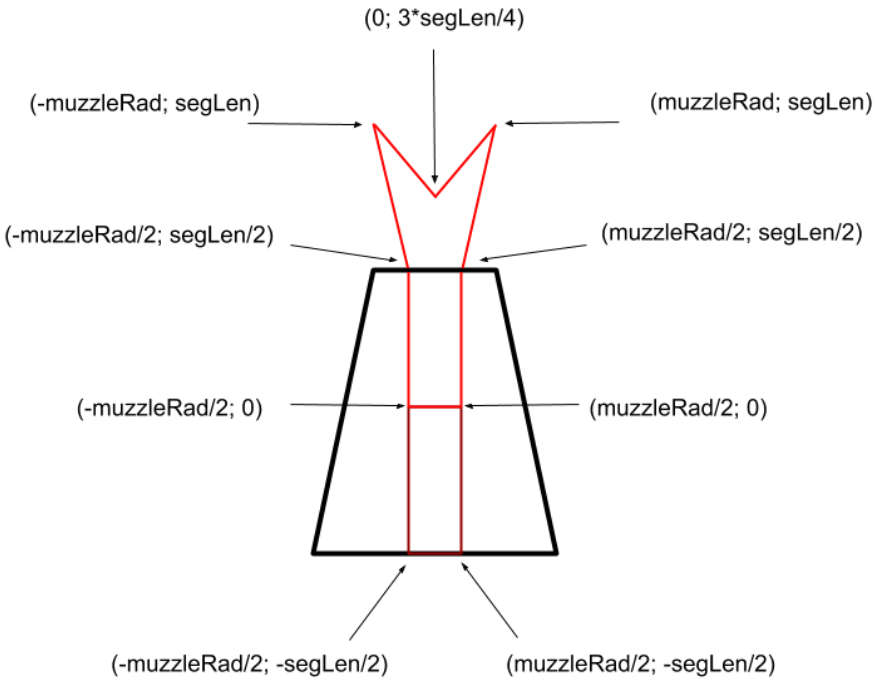


Fig. 6 - Tongue constuction

BODY SEGMENT

The main element of each bodySegment is a **pivot**. The first pivot is linked to the pivotSnake, all the others are linked together.
The pivot chain represents the snake's skeleton that is not visible. The visible part is made up with **cylinders** or **torus**. The latter has been used in order to achieve a smooth movement during the rotations (see Snake Movement chapter).
To represent the skin more natural during the creation of a new body segment the texture is randomly picked from 4 different textures (taken from the same seamless one).

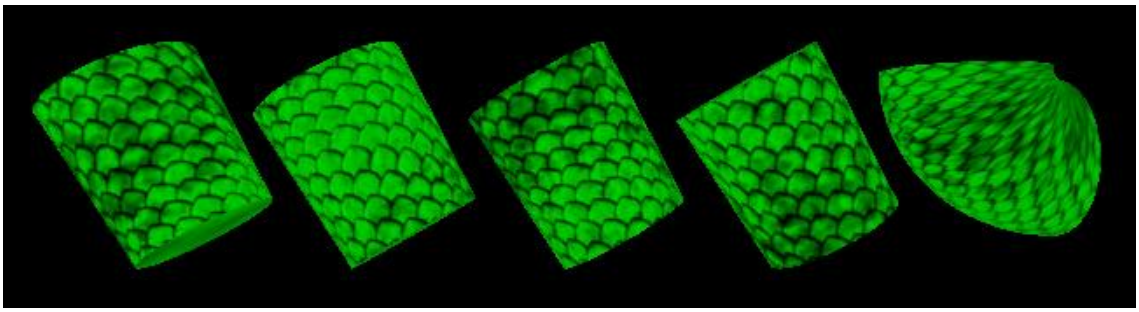


Fig. 7 - Possible body segments with texture

TAIL

The tail has been reproduced using a **cone** shape. It needs to be rotated around two different vertical axes: one on the left and one on the right of the base. It has been realized using two independent **pivots**. More details are provided in the Snake Movement chapter.

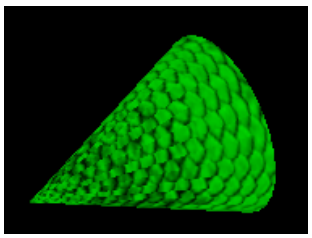


Fig. 8 - Tail wih texture

APPLE

The main geometry of the **apple** is a torus with the radius (*appleRadius*) equal to half tube (*appleTube*).
The **petiole** geometry is represented by a cone and is added to the apple.

The apple, according to the method randPosApple(), will spawn in a random position. The same method will be called every time the apple is eaten.

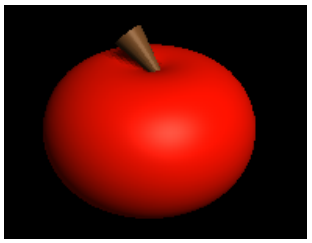


Fig. 9 - Apple

CONSTANT MOVEMENTS

The time variable **t** is used to control the movement of the world (i.e. all the objects except the snake's body segments).

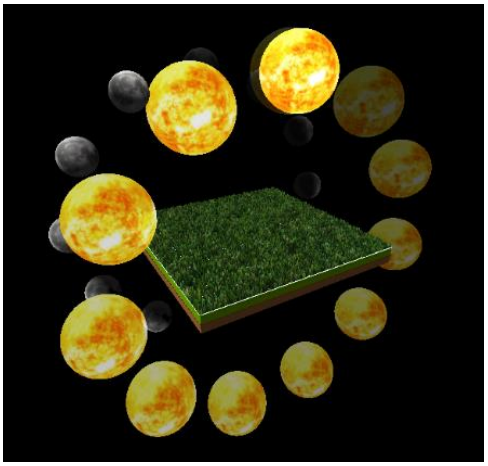


Fig. 10 - Sun & Moon movements

It is used to determine:

- The coordinates of the Moon and the Sun;
- The rotation of the background;
- The horizontal motion of the tongue;
- The rotation of the snake's head to emulate a bite series;
- The rotation and the vertical motion of the apple.

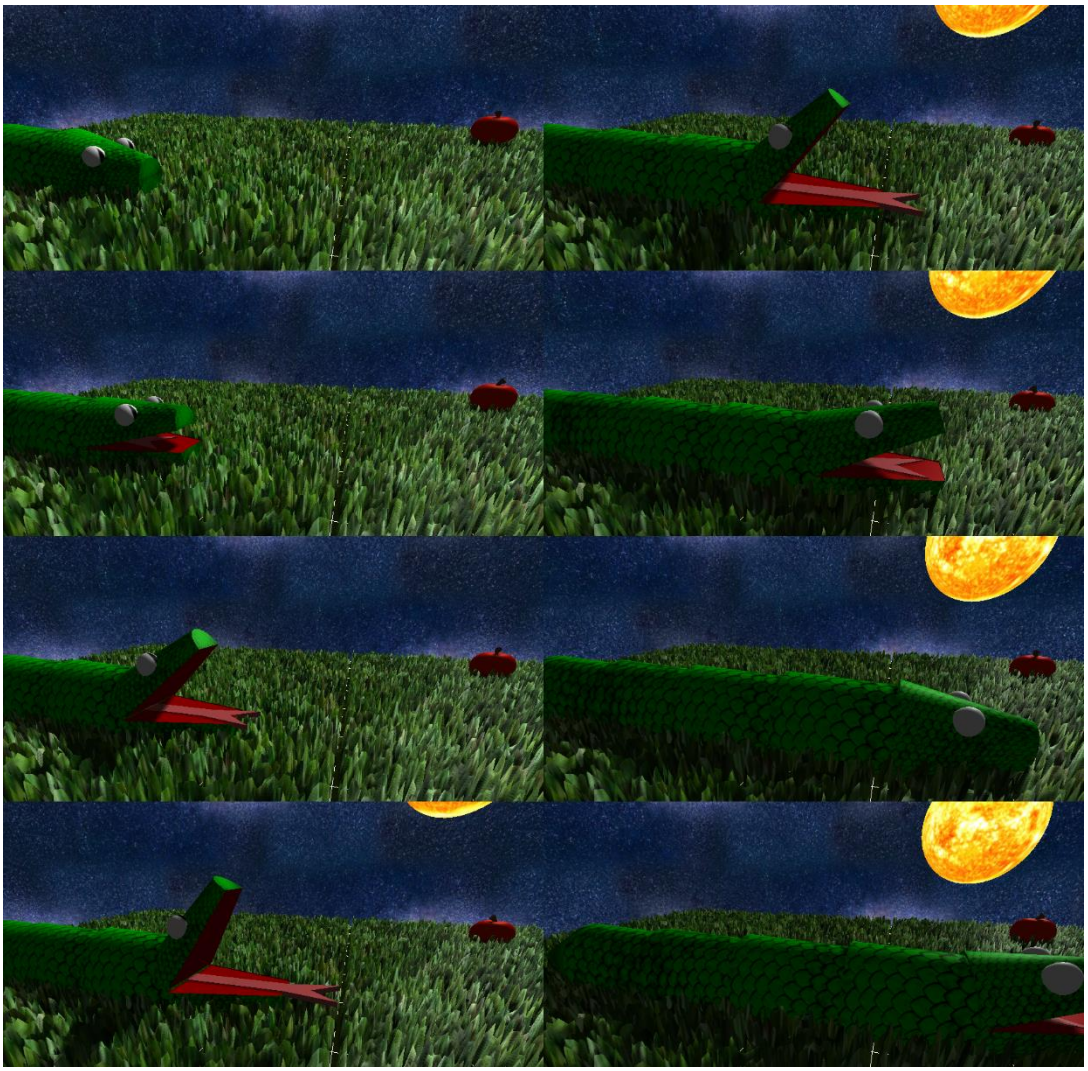


Fig. 11 - Movements in game

SNAKE MOVEMENT

The snake's body is composed by a "**skeleton**" of pivots and the "**skin**" visible to the user. Each segment of the skin is added as child to the corresponding pivot and they moves in different ways.
Apparently the snake slithers on the grass with a continuous and smooth movement, but actually a **timer** rules the snake movement.
The initial timer value is **t_steps** and at the end of each iteration it's decrease by 1. Varying t_steps varies the snake's velocity.
The user can change the snake's direction with the directional arrows, the selected direction, if reasonable with the current direction, is stored in a buffer (called **dirBuffer**). The use of a buffer allows to the user to rapidly create a sequence of changes of direction.

PIVOTS' MOVEMENT

The pivots occupy the center of the cells and move only when the timer reaches the zero.
If dirBuffer is empty the snake goes straight on the current direction, otherwise it computes if the **turn** is on the Left or on the Right.
As showed in the figure, each direction has a numerical code.

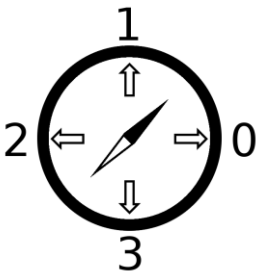
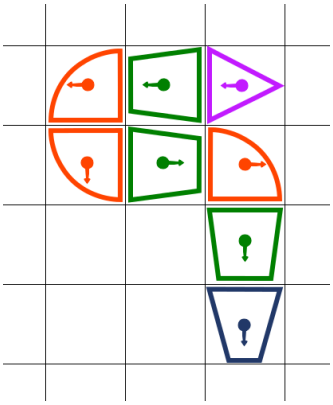


Fig. 12 - Direction coding

turn = dir - prevDir		prevDir			
		RIGHT (0)	UP (1)	LEFT (2)	DOWN (3)
dir	RIGHT (0)	**0**	**-1**	N.A.	**-3**
	UP (1)	**1**	**0**	**-1**	N.A.
	LEFT (2)	N.A.	**1**	**0**	**-1**
	DOWN (3)	**3**	N.A.	**1**	**0**

Rectifying the -3 with 1 and 3 with -1 is possible to identify a **left turn** with -1 and a **right turn** with 1.
With the turn variable is possible to rotate the pivotSnake of ±90° or 0°.
Since the snake's body follows the head's movement, a FIFO memory (called **turnList**) is implemented to store the previous turns and to rotate correctly the rest of the body.

EXAMPLE



segment	head	body0	body1	body2	body3	body4	body5	tail
direction	DOWN	DOWN	RIGHT	RIGHT	DOWN	LEFT	LEFT	LEFT
coded direction	3	3	0	0	3	2	2	2
difference		0	-3	0	3	-1	0	0
turnList		0	1	0	-1	-1	0	0

Fig. 13 - Example of snake configuration with direction study

SKIN'S MOVEMENT

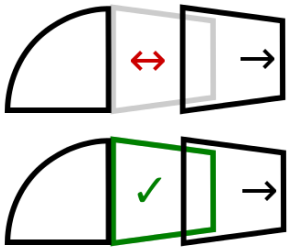
As already said, each skin segment is connected to the respective skeleton pivot.
By default, the skin's RF is centred in the pivot (there isn't any translation).
To allow the motion while the wait of the timer, the skin is translated.
To allow a smooth slithering, when the timer ends the skin should be at the same position of the default position of the next step.

$$d_y = step_{lenght}\left(1 - \frac{timer}{t_{steps}}\right)$$

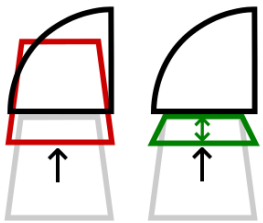
Only the cylinders (straight segments) can be translated, the torus (turn segments) cannot do it, because the motion of the skin starts and finishes in different axis.

There are three problems to solve:

- Cylinder before the torus: the translation of the segment would create a disjunction the two objects, so a **support segment** (fxed at the torus) is added to keep the snake intact.



- Cylinder after the torus: the segment would penetrate the torus, so it is also **scaled** in order to fix the front side with the torus and move just the back side.



- Tail after the torus: scaling the tail is not suitable, the only solution is to **rotate** the tail reducing simultaneously the radius of the torus. The angle of the torus cannot be reduced programmatically, so during the startup of the system, a list of 10 tours with different radius is created and during the movement the torus is **replaced** by them.

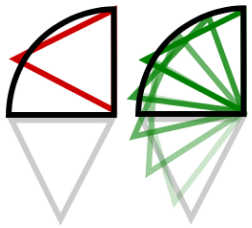


Fig. 14 - Error/Solution given by the skin movement

Another problem is represented by the action reactivity: if the skin doesn't move itself the user understand that he must wait the end of the timer to see his action executed; but a continuous motion implies also a continuous action reactivity.

To solve this problem, the timer has been divided in 3 intervals:

1. During the first interval the snake's head starts to occupy the next cell, but it's mostly in the previous one, so the action is considered still valid for the previous line and it's executed immediately;
2. During the second interval the snake's head cannot be considered in the previous cell anymore, but immediately executing the action would lead to a jump, so in this interval the action is just stored in the dirBuffer;
3. During the third interval the head is mostly in the new cell, so if the user wants to turn, it's reasonable to do immediately.

The first and third interval of two consecutive steps imply the same action, so the reactions at a user input are two: execute the action immediately or wait that the snake reaches a reasonable position to execute it.

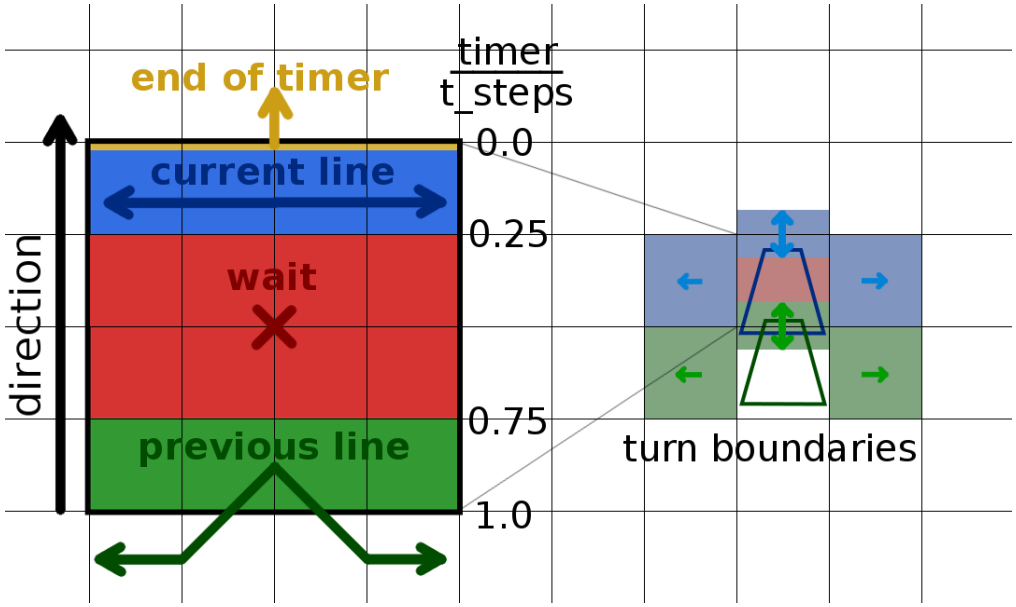


Fig. 15 – Step time division for action reactivity

The action reactivity implies that the skeleton may be moved also when the timer hasn't reached the 0, in these cases the skin motion is restarted with new position.

After the execution of the action the snake's head can be:

- in an empty cell: nothing happens;
- in an invalid cell (occupied by its body or out of the world): in this case the game ends and the program's state changes (see Transition System chapter). To determine if the snake bites itself, a FIFO memory, similar to the turnList, is used to store all the cells visited from the snake's head; the oldest position is discarded every new pivot motion.
- in the apple cell: the snake grown: all the lists used to store the snake information are extended by 1, the tail is replaced with a new cylinder and added again as child of the new segment.

CAMERA

The game can be played with three different views; the user can change it pressing the ‘v’ key.

1. **Perspective**: in the default view the camera is located at one side of the world and it looks at the ground’s middle point;

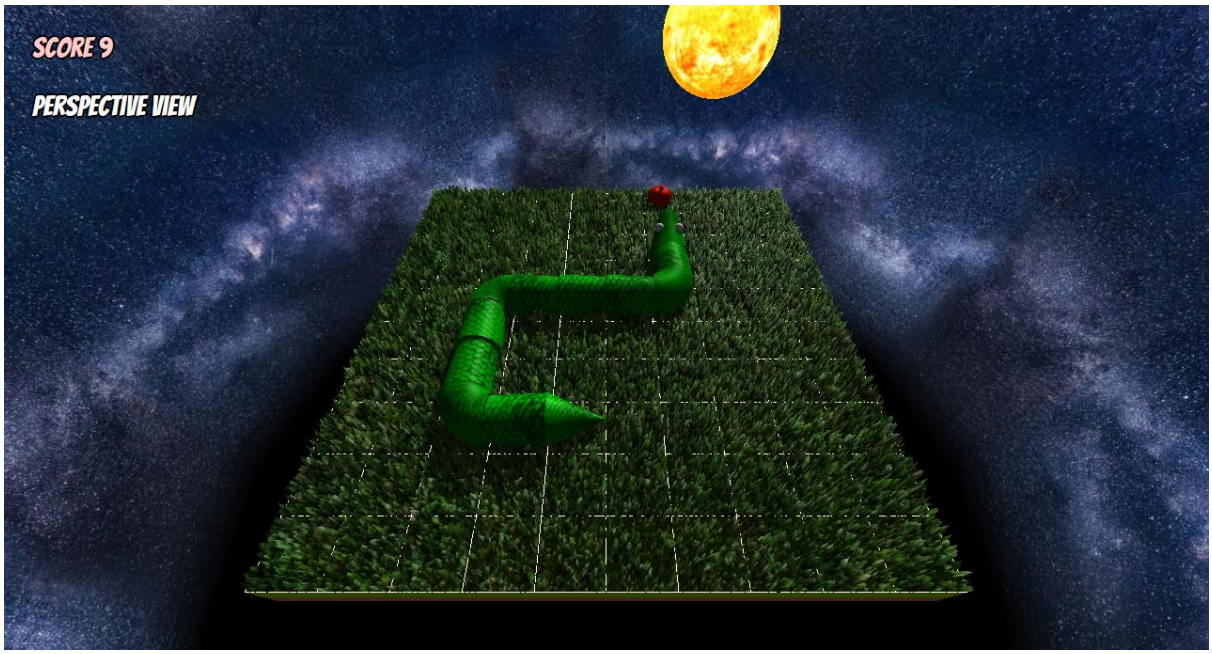


Fig. 16 - Perspective View

2. **Top**: the second view is obtained positioning the camera along the z axis of the scene and looking down at the ground’s middle point;

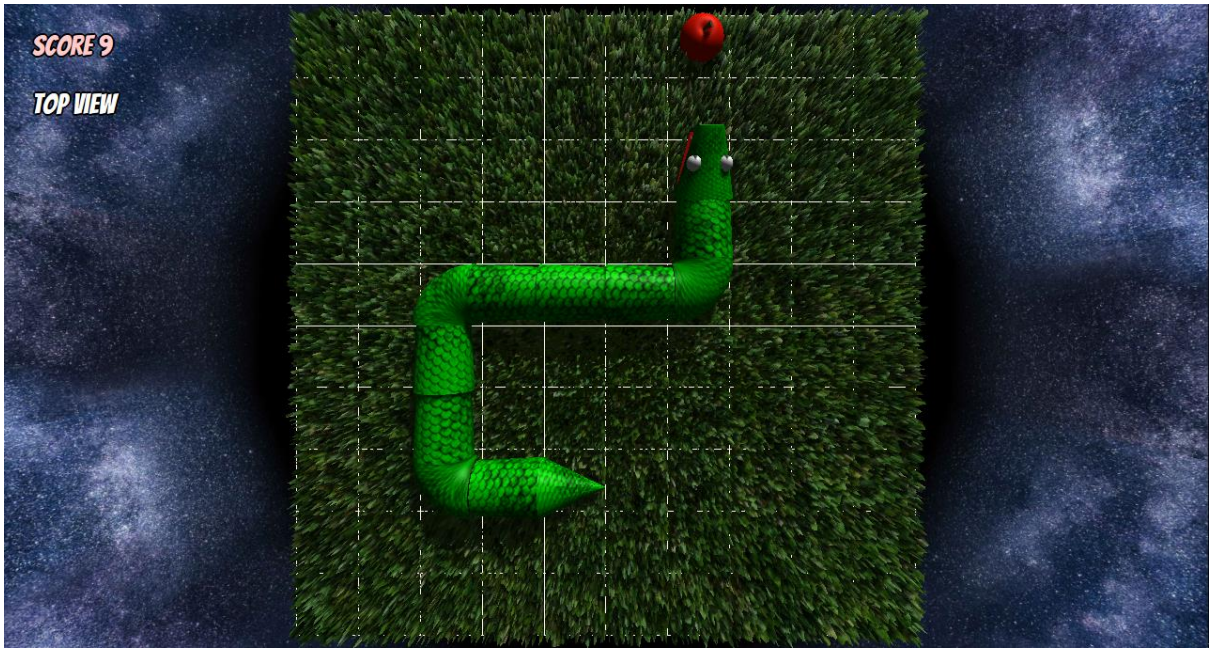


Fig. 17 - Top View

3. **Point of View**: in the last view the camera follows the snake and shows to the user the snake's head and the world in front of it. The camera cannot be added as child of an object, so every iteration the camera is moved (also the view point is computed).

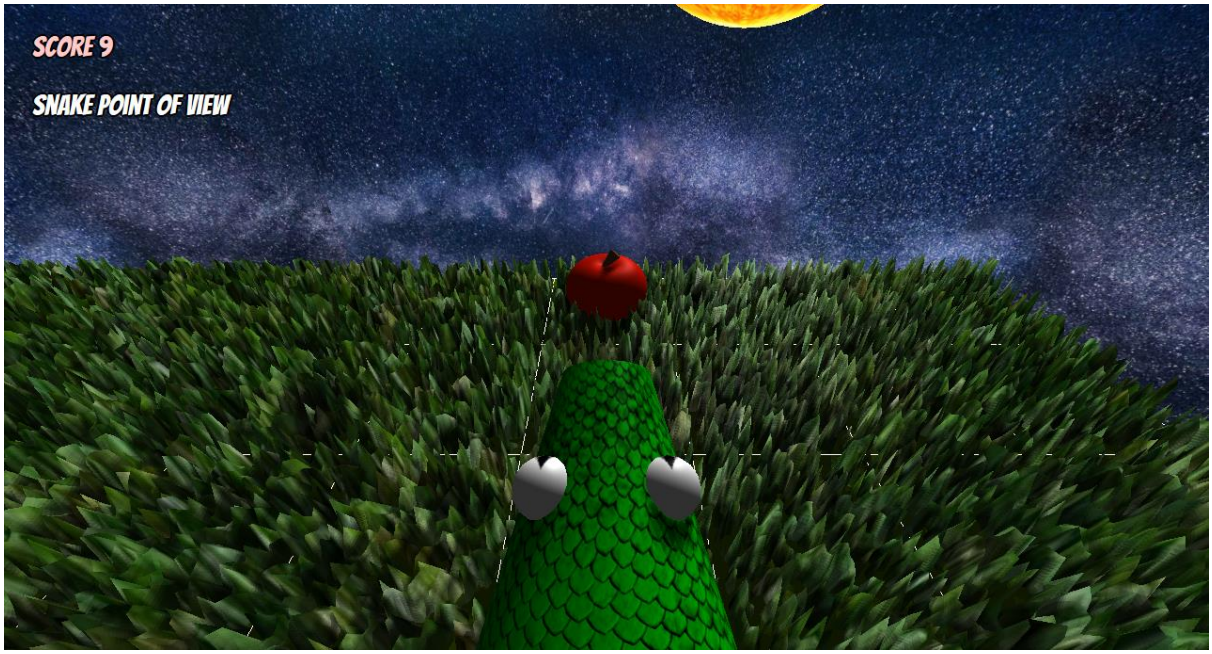


Fig. 18 - Point of View

SOUNDS

When the game is launched a background music [4] starts. The user can pause it and then start it again pressing the 'm' key.
During the game there are other sounds:

- A juicy bite [5], reproduced whenever the snake eats the apple;
- ~~A PAC-MAN game over effect used with the same scope.~~
- A scream[1], reproduced when the game is over, so when the snake bites itself or goes out of the world.

[1] The scream has been done by a member of the group.

TRANSITION SYSTEM

An initial menu introduces the user to the game commands. The user can:

- **start** the game pressing *enter*
- **move** the snake using the *directional arrows*
- **change the view** [prospective - vertical - POV] pressing *v*
- **turn the music on/off** pressing *m*

When the game starts, the initial menu is hidden and an **info box** is showed. It contains the currents score and view. The score colour changes from white to red w.r.t. the current score.
When the snake dies, a new box pops up, displaying the score achieved; if the user wants to restart the game, he only needs to press *enter*.
This process is handled by the function change_state() according to the following flowchart:

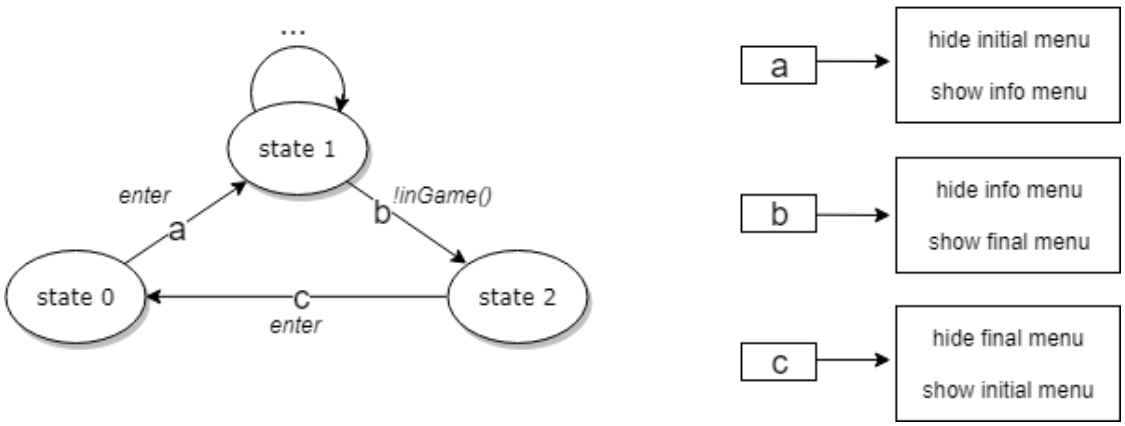


Fig. 19 - Transition system of the game

The state 0 and 2 are only listeners that wait for the *enter* command.
The state 1, in addition to check if a game is over, has to manage the movement inputs according to the view mode, as discussed in the Snake Movement chapter.

SITOGRAPHY

[1] [Online]. Available: https://threejs.org/.

[2] [Online]. Available: https://www.pexels.com/photo/starry-night-sky-1205301/.

[3] [Online]. Available: https://www.solarsystemscope.com/textures/.

[4] [Online]. Available: https://www.hooksounds.com/royalty-free-music/curious/31569/.

[5] [Online]. Available: https://www.youtube.com/watch?v=FDi6GT3Jgf8.

APPENDIX – OBJECTS' LIST

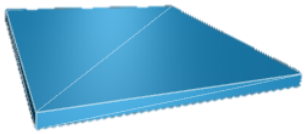
All the following images have been created on ThreeJS website [1].

BOX

Used in:

- Ground
- Mud
- Background

THREE.BoxGeometry (width, height, depth)

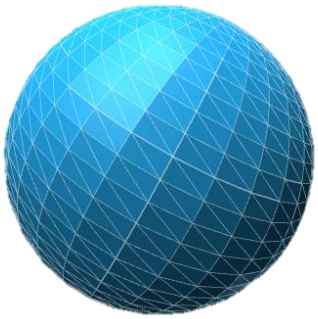


SPHERE

Used in:

- Sun
- Moon
- Eyes

THREE.SphereGeometry (radius, widthSegments, heightSegments)

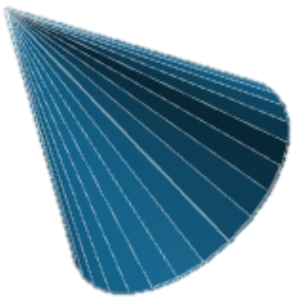


CONE

Used in:

- Tail
- Petiole

THREE.ConeGeometry (radius, height, radialSegments)

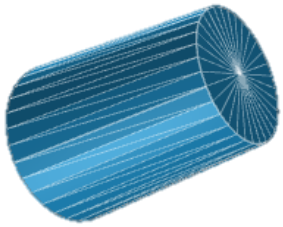


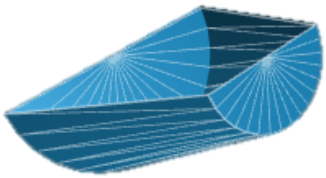
CYLINDER

Used in:

- Straight body segment
- Chin & Head

THREE.CylinderGeometry (radiusTop, radiusBottom, hiegth, radialSegments, thetaLength)



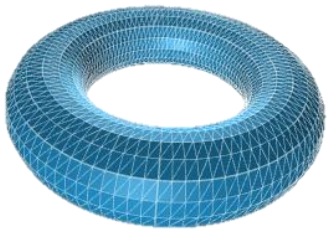
thetaLength < 2π


TORUS

Used in:

- Curved body segment
- Apple

THREE.TorusGeometry (radius, tube, radialSegments, tubularSegments, arc)



tube > radius:	arc < 2π