A Report on the Anatomy Atlas and its features

Daniel Tauritis, Waldemar Zeitler, Hannah Kathmann,
Marcel Merten, Patrick Plate, Dustin Augsten
Andy Augsten, Arian Mehrfard, Thomas Tannous,
Florian Rohde, Dominik Sauer, Daniel Albensoeder
CGVR
Bremen University

CONTENTS

A Report on the Anatomy Atlas and its features

***Abstract*—This document provides inside into the technical details of the VR Anatomy Atlas, giving an overview of the implementation and configuration. The input device was selected to grant an immersive experience in virtual reality.**
Our Software allows users to interact with a human body in a virtual operating room and learn from it.

I. Introduction

This technical report introduces the reader into our implementation of the anatomy atlas, which was developed during the summer semester 2017 as part of the bachelor's project "VR Assembly" of the University of Bremen. It is expected that the reader has a basic understanding of the Unreal Engine and working with Blueprints. It is also a virtual reality project, which is designed for a head mounted display in combination with the Vive Motion controllers.

II. Problem definition and features

A. Center Of Mass

All organs have their pivot point at the feet of the body but for proper snapping we need the actors real location. Hence we need the center of mass of each organ which becomes the new location of the actor while we subtract it from the initial position to fix the offset of the geometry.
Due to complex collision the center of mass calculation included in Unreal Engine did not work properly. Our workaround was to use a custom calculation which however only worked in the Editor and not in the packaged project. We extracted the locations in Editor mode and stored the values into a datatable so that we can use these in our project.

B. Snapping

Snapping point detection is realized in the Tick event of the organs. Here it checks if the organs location and rotation are near the initial values stored earlier in the BeginPlay event. If true, the boolean FoundSnap is set to true, otherwise to false.

C. Ghosting

Inside the Tick event if FoundSnap is set to true we spawn the actor OrganHologram and promote it to a variable Ghost. It copies the transformation and the mesh geometry of the attached organ. The texture is a transparent green material. The actor is being destroyed if FoundSnap is set to false or if no organ is attached.

D. Hiding Organs

When firing the grab trigger of the left controller the user can move through organs to hide them. For that purpose a boolean is set to true when firing the trigger and in the Tick event of the left controller it checks if the boolean is set to true and disables all overlapping Organ actors and calls the function DoRumble which gives haptic feedback on the controller.

E. Skin

When the application is being started, the humans body is being slipped by a skin. Before the user can grab an organ he first has to remove the skin by overlapping it and firing the grab trigger of the right controller. The logic of the skin removal is in the HideSkin function which checks if the skin is slipping the body. If true, it checks for overlapping and replaces the skin material of the body with a transparent material. The function returns either true or false, true meaning that the skin was already removed before and the caller function which is the actual Grab function, can grab an organ and false meaning that the skin was not removed due to no overlapping with the skin or that the skin was just removed and another call is now required to grab an organ.

F. Scaling operation bed

The user can scale the operation bed up and down by using the trackpad of the right controller. Here the function ScaleOPBed is being called which changes the transformation of the bed based on the passed ScaleValue parameter and a new Z coordinate is being calculated for the human body based on the scale of the bed.

G. Movable Furnitures

The user can move furnitures using the left hand. For that purpose we have created a new Actor class MoveableFurniture. When the BeginPlay event is fired, it promotes the instance of the left controller and its initial position into variables. The initial position is later being used for the Reset feature.
The whole logic is inside the Tick event. Here it checks if the boolean Attached is set to true. If not, which means that this furniture is not being hold, it checks if the controller is currently grabbing, colliding with the furniture and if no other furniture is being hold at this moment. If all this results into true, the X, Y and Z coordinates of the furniture are stored into variables, whereby the X and Y coordinates of the grab sphere are being subtracted from the ones of the furniture. The boolean Attached is being set to true and for the controller we remember that it is currently holding a furniture.
For the next ticks the boolean Attached is set to true until the controller is not grabbing anymore because then the booleans Attached and Holding are set back to false. With each tick we

update the location of the furniture currently being grabbed, giving it the coordinates we stored earlier and adding the X and Y coordinates which we extract from the location of the grab sphere. Thus resulting in the furniture moving smoothly with the hand.

H. Reset

The Reset function resets the application to its initial state. It resets the location of the human body, all its organs if no organ is attached to the hand at the moment, the bed and the movable furnitures. It also resets the skin material of the body and the values of several variables.

I. Medicine posters

We have included over 30 medicine posters in our application. The user can switch between these by pressing a button. For that purpose a function is being called on the Poster actor which simply changes the sprite to the next one of a list.

J. Tooltips

Tooltips are stored as String values in a XML file where the tooltip for each object is tagged by its name. Reading and parsing the XML file is being done in the Tooltip.cpp class which inherits from AActor class so that it can be spawned to create an instance. The specific object name and the name of the XML file have to be passed as argument to the Load function which is declared as UFUNCTION so that it can be called from Blueprints. The parsed String gets stored in the variable Text which is declared as UPROPERTY.
The widget TooltipWidget reads the value from the variable Text to print it. Since we can't properly spawn widgets on viewport in Virtual Reality we have created another actor Tooltip which simply implements the widget as component and can now be spawned in the world. The position of the tooltip is being updated with every Tick event where it copies the location and rotation of the camera and moves a little bit forward which is the same procedure as for the third person character in Unreal Engine.
The user can show tooltips for an organ by pressing a button while holding it. The tooltip then gets intialized if not already happened and is being spawned in the world and gets removed if the user releases the tooltip button or the organ itself.

K. Menu

The menu is designed in a widget which is then stored in a widget component in a new actor: the "MainMenu3D" actor.

The "MainMenu3D" actor is placed in the world and is acting as a spawner for the menu. At "BeginPlay" it will save the left and right hand of the player in variables and add a widget interaction component to the right hand so the player can interact with the menu. In the "Tick" event there are three steps to be done:
1. The position and rotation of the menu is constantly updated so that it stays on the players hand and is facing towards him
2. Check whether the players hand is facing up or down and then set the visibility of the menu to the appropriate state to make it visible or hide it.
3. Draw a forward line on the right hand to make interaction with the menu easier whenever the menu is set to visible. As the line is implemented as debug line it will only show in editor mode or if the project is packaged in development or debug state. The interaction however will work nonetheless if the project is packaged in shipping. For easier interaction one could then touch the menu items.
The main menu widget contains the graphics and the functionality for the menu items. There are two graphics: one if the item is selected by the user and one if not. On top of the graphics there is a text component with the name of the item. A progress bar indicates the click event for an item. When it is filled the appropriate item is clicked. This is done by the hovered/unhovered events for the items. When the player hovers over an item a timer starts that calls a function for that item after a given time. If the player unhovers the item in the meantime the previous set timer is canceled.

L. Left-Handed Mode

The left-handed mode can be toggled in the menu. A check box indicates the state. When the player toggles the mode a function is called which checks the current state and mirros the hand meshes if it was unchecked before or sets the meshes back to normal if it was checked before. The player can easily see that the meshes were mirrored and just needs to switch the controller afterwards.

M. Surface Collision Actor

Users can grab and move organs using their right controller. This requires a system determining what organ can currently be grabbed by the user.
Originally grabbable organs were determined using a sphere collision around the player controller. This allowed users to grab any organ, even those encased by other organs. Such organs would not be accessible during an actual surgery; The occluding organs would need to be removed first.
The SurfaceCollisionActor collides with objects in its surroundings. It detects collisions along the surfaces of objects, and cannot collide with objects surrounded by other collidable objects.
The SurfaceCollisionActor contains the following components:

- CollisionSphere, a small, movable collision sphere that collides with other objects.
- CollisionSphereRadius, a static collision sphere that is attached to CollisionSphere. Its radius is slightly larger, overlapping with objects and making the collision detection more lenient.
- StaticSphere, a large, static collision sphere and the root of the SurfaceCollisionActor. It also overlaps with objects.
- A physics constraint constraining the CollisionSphere to the StaticSphere's position.

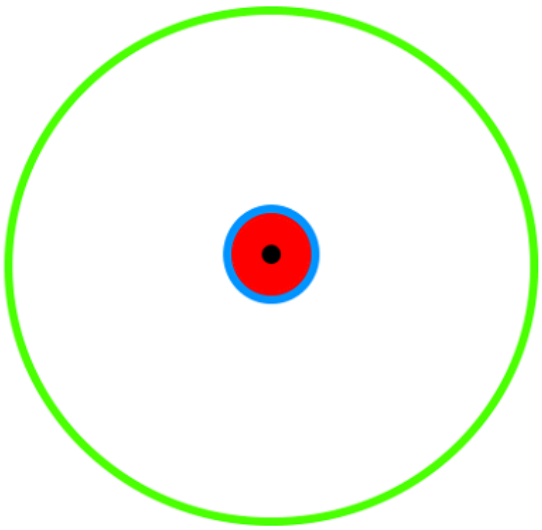

Fig. 1. Surface Collision Actor I

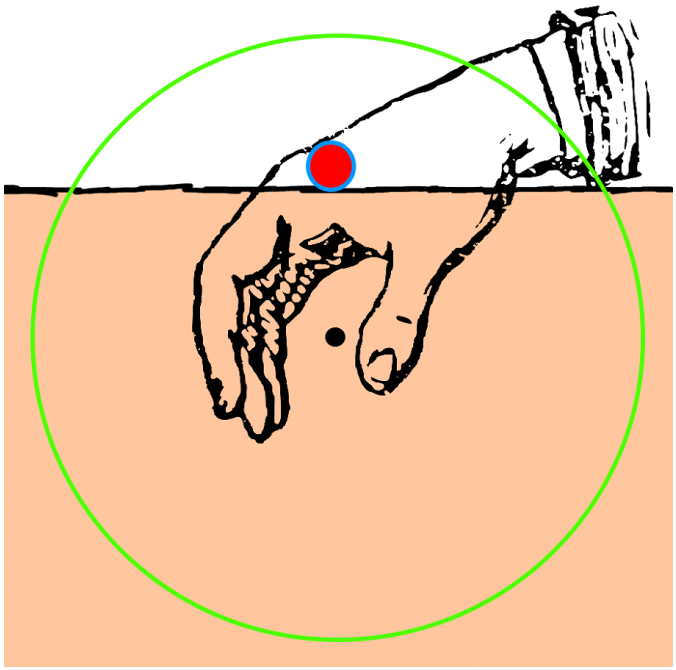

Fig. 2. Surface Collision Actor II

The SurfaceCollisionActor reports a collision when an objects overlaps with both the CollisionSphereRadius and the StaticSphere. Other actors can inquire about collisions using SurfaceCollisionActor's functions. Actors requesting a single colliding actor always receive the newest colliding actor. Otherwise large objects that collide with the SurfaceCollisionActor for a longer period of time would prevent smaller objects from being selected.

We use the SurfaceCollisionActor as a component of RightControllerPawn, positioned between the mesh's thumb and index finger. The CollisionSphere is visualized using a transparent mesh, giving the user feedback about its location. The RightControllerPawn requests the most recent colliding Actor from the SurfaceCollisionActor. This lets it determine what organ can currently be grabbed.

When the SurfaceCollisionActor touches the surface of an object, the object will overlap both the StaticSphere and the CollisionSphere radius, selecting it.
If the controller is moved further into the object, the CollisionSphere remains at the object's surface. Only objects on the surface can still be selected.
Should the user move the controller deeper, the objects on the surface may be too far away to overlap with the StaticSphere. No objects will be selected.
The physics constraint will attemot to make the CollisionSphere follow the StaticSphere. If no path can be found and the distance between the spheres reaches a threshold, the physics constraint will 'teleport' the CollisionSphere, forcing it closer to the StaticSphere. This prevents the CollisionSphere from getting stuck on the surface of an organ if, for example, the user swipes their controller across the body.
The SurfaceCollisionActor is flexible and can be used in other context. Parameters like the sphere radii can be customized while or after spawning SurfaceCollisionActor instances. The values of the physics constrant are not all publically accessible, but can easily be edited within the SurfaceCollisionActor itself.
The SurfaceCollisionActor works best if its parameters are set correctly. Its current settings have not been extensively tested, and more testing could reveal if adjusting its parameters require more tweaking.

III. Conclusion and Future Work

We have achieved all our goals for the project. At this moment there are no known bugs, everything is working as expected and no planned features are missing.
An idea which came up when there was not enough time anymore to implement the feature is a multiplayer mode. This would allow a second person to join the virtual world to watch the first person working with the body.