Collision Detection and Teamcenter Haptics: CATCH

Group: May 14-30
Anthony Alleven
James Erickson
Paul Uhing
Logan Scott
Matt Mayer

Introduction

The field of virtual reality is advancing in leaps and bounds. One blocking factor to virtual reality's progression is a lack of reality. Users need to have some feedback on their actions. Haptic feedback is one way to provide users with feedback. An application that provides users with haptic feedback for their actions in the virtual world is needed.

While Dr. Vance's group has already demonstrated the ability to solve the above problem, their solution is not commercially viable. Our project is to make this project more commercially viable by utilizing commercial software.

CATCH is a prototype standalone service to demonstrate manipulation of 3D models using haptics and Teamcenter Visualization. This program will perform collision detection and 3D transformation on object models. This project is important because this system has been previously created using in-house software not suited to commercial use. This project seeks to create a commercial-compatible solution to the problem stated above.

Requirements

Functional requirements

1. Manipulate a cursor in Teamcenter with a haptic device
2. Select an object in Teamcenter Visualization with the haptic device
3. Detect collisions between objects and perform appropriate haptic feedback
4. Support loading part geometry via Jupiter Tessellation (JT) files

Non-Functional requirements

1. The lag time between input and output shall be less than 200ms
2. All public modules and functions shall be documented to the extent at which they could be recreated by a third party.
3. After accounting for lag time, all object models shall be synchronized.

Definition of Terms and Acronyms

Term / Acronym	**Definition**
CATCH	Collision Detection and Teamcenter Haptics
JT	A data format for 3D models developed by Siemens PLM Software
Jttk	JT Open Toolkit, the software being used to interface with JT files.
IPSI	Interactive Physics Simulation Interface, the physics engine produced by Haption to be used for collision detection.
TCVis	Teamcenter Visualization, the tool produced by Siemens PLM Software that will be used to display 3D model representations..
VisController	Teamcenter Visualization Controller, the API that controls interaction w TCVis

Functional Decomposition

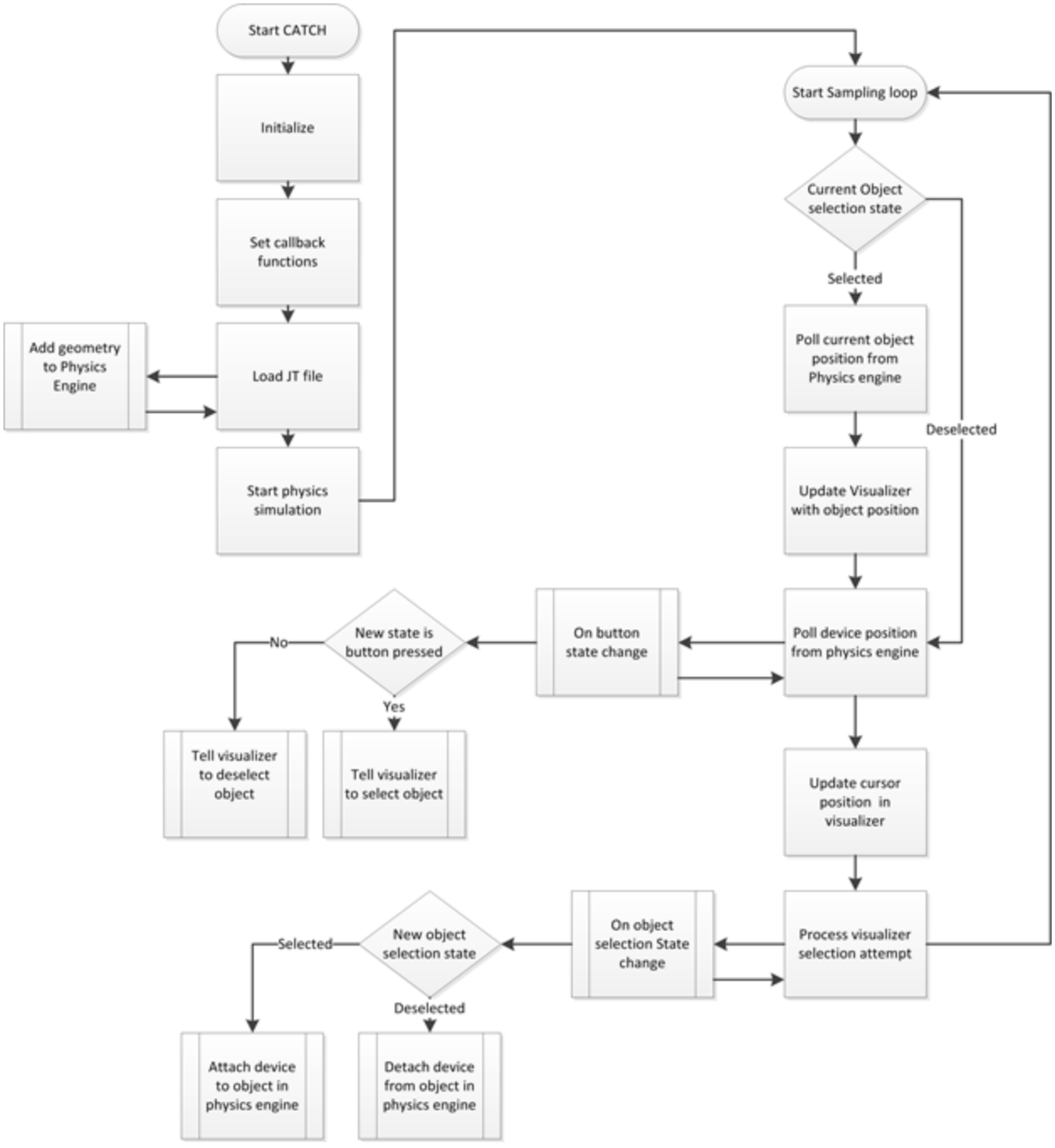

Figure 1.1: Activity Diagram of CATCH

Figure 1.1 describes the state transitions of the software without breaking the system into modules. Note that the first part of the diagram is the initialization code. Most of the code listed above takes place in the CatContext and CatCursor modules, shown in Figure 1.2.

System Analysis

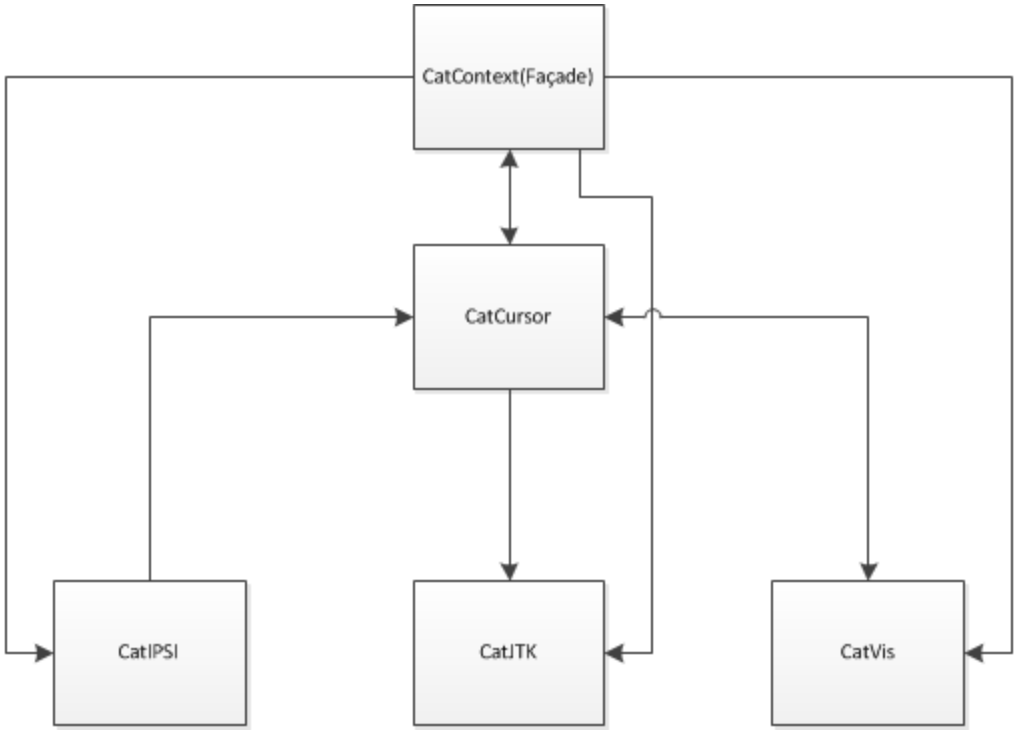

Figure 1.2: Breakdown of modules

This project can be broken down into five modules. The CatContext module acts as the facade. It is how other programs will interact with our modules. CatCursor controls information flow between the other modules. CatIPSI will interact with IPSI, the physics engine for this project, and Virtuose, the hardware. CatJTK interacts with a local assembly file using JT Open Toolkit. CatVis interacts with Teamcenter Visualization.

Module Descriptions

Module Breakdown

- CatContext(Façade)
    - Initializes all other modules.
    - Starts the Main loop of program.

- CatCursor
    - Polls for cursor position and sends the update to VisController.
    - Polls for button status for object selection logic.

- CatVis
    - Initializes the properties of this module.
    - Tells TCVis what file to open.
    - Sends object and cursor transforms to TCVis.
    - Sends back item selection data.

- CatJTK
    - Initialize virtual scene in IPSI and TCVis.
    - Update JT scene positions for saving and loading scenes during operation.

- CatIPSI
    - Runs physics engine for collision detection and haptic feedback.
    - Integrates with haptic device or other input devices.
    - Generates Transformation matrices for CatVis.

Software Design

This project uses a facade design schema. A facade design schema uses a single class to abstract away details from member classes included in it by combining various member functions into larger more powerful functions. For example, a general init() function will contain calls to each member class's init() function. This allows for cleaner code design and better readability of code. The cursor module shown below will do most of the managing of updates and function calls. The primary way the modules will interact will be through callback functions.

Detailed Design

Input/Output Specs.
Input: Transformation matrices from the haptic device, button state, and part selection from Teamcenter Visualization(TCVis)
Output: Transformation matrices after physics calculations to TCVis. Haptic feedback to haptic device.

Exterior Interfaces
Command line interface specifying which file is to be opened.
Input and feedback to and from the haptic arm.

Hardware Specs.
In addition to the desktop environment that CATCH will be running on, CATCH will be interacting with an external haptic device. The haptic device is a Virtuose 6D35-45 haptic arm made by Haption S.A.

Software Specs.
CATCH will be programmed using C++ and must be able to compile in Microsoft Visual Studio 2010 for use in a Windows environment. TCVis will be used as the visualization software for the simulation, IPSI is the physics engine and interacts with the haptic device, and JT Open Toolkit initializes the simulation scene.

Test Specs.
This system will be tested by successive sets of demo programs. The system of modules will be used as a set library of functions that interacts with different devices. This makes a demo program method the best way to test the functionality of our software. There will be three demos each increasing in complexity:

1. Create a demo to interact with a Virtuose simulator and our current Teamcenter Visualization Controller Demo.
2. Create a demo to interact with the Virtuose simulator and Teamcenter Visualization.
3. Create a demo program to interact with Virtuose and Teamcenter Visualization. Consider this the final project demo.

A test will be defined to be successful if the software is able to track cursor movement, select parts, translate parts, detect collisions, and produce realistic force feedback. If these are not met, both the demo code and the modules will be checked for errors in order to ensure that the demo module is not the source of the errors.

Simulation/Model

We have access to a virtuose simulator allowing us to emulate input from the physical device. The models that we will be manipulating are provided by jt files and read by JT Open Toolkit.

Prototyping/Testing

The prototyping will get successively more advanced as more advanced demos are introduced. The list of demos for test purposes and prototyping for the client to see project progress are listed above in Test Specs.
