TEST DOCUMENT

SKETCHUP BLOCKS - ANDREW SMITH - CSIR

Version 1.3

13/10/2013

11081092 - ET, van Zyl

11211548 - WH, Oldewage

11183153 - JL, Coetzee

CONTENTS

CHANGE LOG

Version 1.0	25/08/2013	Document created.
Version 1.2	15/09/2013	Added Technical Spec reference.
Version 1.3	13/10/2013	Revised document

1. INTRODUCTION

1.1 PURPOSE

The purpose of this document is to give an overview of the tests performed on the Sketchup Blocks project and the overall test coverage provided by these tests. The quality assurance activities performed are also outlined.

1.2 PROJECT SCOPE

Sketchup Blocks aims to provide the client with a system that will convert physical 3D models into digital form. The system will include software and hardware components.

Hardware components:

- The physical blocks with fiducial markers attached (called 'smart blocks').
- The construction floor on which the smart block constructions are built.
- Approximately five web cameras.

The scope of the software components will broaden from release to release. The client only requires the first release to consider the project successful. The releases following the first release will address additional concerns and implement extra features mentioned by the client.

Release 1:

The initial system will be able to recognize a small set of geometrically simple objects without any time constraints. After the blocks have been transformed into a digital model, the vertices of the model will be output to an object file for import into Google Sketchup.

Release 2:

This release aims to limit the time taken to recognize a fiducial and recreate the model it represents. The amount of constructed and recognizable blocks will also be increased. The setup of the system will be simplified to improve mobility and allow for fast setup. The removal of blocks will be detected without extra input from the user.

Release 3

This release aims to add a viewing window that will display the model as it being created thereby providing visualization before exporting to a model. The detection of the fiducials should be possible in suboptimal lighting conditions.

Optional release

Custom blocks may be imported to the smart block database. Users can view the construction of other users over the network.

1.3 RELATED DOCUMENTS

Project Vision and Scope (Sketchup Blocks project, by CICO).
Design Specification (Sketchup Blocks project, by CICO).
Architectural Requirements (Sketchup Blocks project, by CICO).
Technical Specification (Sketchup Blocks project, by CICO).
Project Management (Sketchup Blocks project, by CICO).
Sketchup Blocks Developer Blog (http://sketchupblocks.wordpress.com/).

2. SYSTEM DESCRIPTION

The project subscribes to the concept of the Internet of Things. The Internet of Things is centred around the idea of 'smart objects' which are physical objects and devices that are given a virtual representation and linked to information networks. These smart objects will ideally be able to capture, react to and communicate about data captured from their physical environment autonomously.

This project aims to create a system that can convert physical 3D models into digital models that can be viewed and edited in Google Sketchup. The physical model will be built with blocks that have fiducial markers attached. The locations of these blocks will be picked up on web cameras by Reactivision software and the resulting data will be used to create digital 3D models that can be viewed and exported to Google Sketchup.

3. QUALITY ASSURANCE PROCESSES

3.1 TEAM PREPARATION AND TRAINING

The development team invested active effort and time to understand the technologies that are employed in the Sketchup Blocks system. These technologies include ReacTIVision and the Processing libraries. Time was also devoted to examining the TUIO protocol and Collada file format.

Proper understanding of the technologies enables the developers to use these technologies comfortably and effectively, thus enabling them to deliver a high quality product.

3.2 STATIC ANALYSIS

The intermediate documents for the Sketchup Blocks system undergo continuous evaluation and revision. The requirements, architectural and function specifications are all examined to ensure completeness and traceability.

It is also ensured that all the high-level documentation, technical documentation and the developer blog remain consistent and up to date as the Sketchup Blocks system evolves. This ensures that the inner workings and logical flow of the system remain clear to the developers, client and any external parties involved in system evaluation.

Such continued clarity, in turn, serves to minimize errors and inconsistencies in the code. It also ensures that the advice and comments from any third parties are useful and applicable.

3.3 CACLULATION CORRECTNESS

The intermediate documents for the Sketchup Blocks system undergo continuous evaluation and revision. The requirements, architectural and function specifications are all examined to ensure completeness and traceability.

It is also ensured that all the high-level documentation, technical documentation and the developer blog remain consistent and up to date as the Sketchup Blocks system evolves. This ensures that the inner workings and logical flow of the system remain clear to the developers, client and any external parties involved in system evaluation.

Such continued clarity, in turn, serves to minimize errors and inconsistencies in the code. It also ensures that the advice and comments from any third parties are useful and applicable.

3.4 EVALUATION OF ITERATIVE METHODS

At the current iteration, the Sketchup Blocks system makes use of Particle Swarm Optimization, but this is temporary. For the final release, the search methods will be chosen according to the search space that they will act upon. It will also be shown that the chosen search method will converge on the correct solution.

3.5 TESTING

Testing is obviously a very important aspect of quality assurance as well as quality control. As is explained in the Architectural Requirements, the Sketchup Blocks system very nearly follows a pipeline process in which many separate components act upon the given input data. It is, thus, of paramount importance that every individual component is subjected to thorough unit testing as well as different levels of integration testing.

Interaction with the Sketchup Blocks system takes place by means of command blocks and not only through mouse/keyboard input. Consequently, thorough testing of the command block-based user interface must also take place.

4. UNIT TESTING

Unit tests are performed continuously and are grouped into component-based categories. JUnit has been chosen as a testing framework. The tests are executed by means of the JUnit plugin for Eclipse.

4.1 UNIT TESTS OF THE PIPELINE COMPONENTS

4.1.1 INTERPRETER

The Interpreter's unit test ensures that the initial configuration of the Interpreter object is correct. It also ensures that TUIO object events are interpreted and passed to the Session Manager correctly. The events fall into one of three categories: add object, remove object and update object and each of these are tested. The Camera Event returned by each of these functions is then examined to ensure that the data is wrapped correctly.

4.1.2 SESSION MANAGER

The Session Manager's unit test ensures that it initializes the Model Constructor, Lobby, Model Viewer and Database correctly. It also ensures that all the necessary Model Change Listeners are registered with the Lobby.

The most important function of the Session Manager is to look up any Camera Events in the Block Database and send this, along with the Camera Event to the Model Constructor. The unit test ensures that the blocks are recognized correctly as being a smart block, user block or command block. It also tests that the Session Manager copes gracefully with invalid block types.

In addition the unit test ensures that the camera position updates are stored and propagated to the Model Viewer correctly.

4.1.3 MODEL CONSTRUCTOR

The Model Constructor's unit test ensures that command blocks that are used for calibrating are sent to the camera calibrator. It also ensures that no calculations are performed for finding block positions before the cameras have been initialized.

Once enough camera events have been fired for a particular block, its position can be calculated. Consequently, unit tests ensure that no position calculation is done before enough events have fired, that calculations are not performed on events that have exceeded their lifetime or have already been used and that calculations are performed when appropriate. The propagation of the smart block and its position to the Model are also tested.

The Model Constructor's unit test also checks that once no more fiducials of a particular block are visible, their removal is propagated to the Model (via the Lobby).

The Model Constructor makes use of many other classes to calculate a block's position correctly. Each of these have their own unit tests.

The Model Center Calculator is tested by providing the necessary input data to calculate the transformation matrix that the model is multiplied by to place it correctly in 3D space.

The inputs are divided into multiple cases, depending on how any fiducials have been observed. The different cases are 2, 3 and more than 3. Each of these test cases are tried by providing input data for which the correct output is known. Note that for this test, the Rotation Matrix Calculator that the Model Center Calculator makes use of, is mocked out. Invalid numerical input data will be picked up by the other classes that the Model Center Calculator makes use of and so, this class is not explicitly tested with invalid data. The test does check that the Model Center Calculator fails gracefully when provided with only one observed fiducial.

The Rotation Matrix Calculator calculates a particular block's transformation matrix. It is tested by providing input data that has been transformed by some known matrix and then comparing the known matrix to the output matrix. Transformations that involve pure translation, pure rotation and various combinations of the two are both tested. It is also tested with invalid data such as non-square matrices. As with the Model Center Calculator invalid numerical data will only be noticed by the mathematical subroutines that this class makes use of. The unit tests for these mathematical subroutines are described in section

4.1.4 LOBBY

The system's lobby handles the local and network access to a given model saved in memory. The lobby also pushes any changes to the model to any connection to the lobby, being local or over the network.

Unit tests would test if any changes to the model reach all connections made with the lobby. All requests made for the model should also be tested to ensure the entire model with all related data associated with the model reach the recipient correctly.

4.1.5 MODEL

The model saves all the latest smart block data from the smart blocks on the construction floor. This data consists of the block model vertices, fiducial data, block position, orientation and debug information.

Tests should be made to ensure that when a block is added to the model, the is block added or updated depending if it already exists in the model or not. Both cases are tested.

Finally a test should be written that when a smart block has been removed from the model that only the data associated with the removed block is removed from the model's data.

A test should be made to ensure that when one gets the model from the model class that the entire model is returned.

4.2 CALIBRATION UNIT TESTS

The cameras and calibration floor were set up and the data provided by ReacTIVision as well as the camera position were recorded.

One test ensures that the calibration class is not calibrated before input is given.

The input to the system is recreated and given as input to the calibration class. After enough data was sent to the class it should be calibrated and The values for the camera position calculated should not differ from the measured position by more than 5 cm per component.

4.3 COLLADA LOADER AND EXPORER UNIT TESTS

These unit tests cover the loading and saving of model data.

4.3.1 COLLADA LOADER

A folder named ColladaTests will exist. It will contain multiple model files in the Collada format. Each of these files will be loaded in a different test. The first file will be a simple untextured cube. The tests folder may also contain more complex models. In addition, the tests folder will contain a file that does not conform fully to the Collada format. This test case expects an exception to be thrown.

4.3.2 EXPORTER

A simple model is given to the exporter along with a filename in the folder ExporterTest. This export should succeed without any errors.

This file is then loaded with the Collada loader and the data from file compared with the file given to export. The file will also be opened in Google Sketchup to ensure compatability and that the file is well-formed.

4.4 MATH LIBRARY UNIT TESTS

The math library includes all the data structures (such as vectors and matrices) that are used as well as the numerical methods such as the Singular Value Decomposer and the LU decomposer.

The elementary algebraic operations for the vector and matrix classes are tested easily by checking that the given input matches the expected output (eg. scalar multiplication, normalization, etc).

All matrix operations are tested with valid input as well as matrices with invalid dimensions. In addition, all operations that require the matrix to conform to certain constraints are tested with valid as well as invalid data. For example, the operation for calculating matrix determinants and inverses is tested with a non-square matrix.

Similarly, the conversion methods are also tested with invalid data. As an example, the method that converts a single row matrix to a vector is tested with a 2 by 2 matrix.

The LU Decomposer is tested by comparing its output to that of Matlab. It is tested with singular matrices and non-square matrices as invalid input. The SVD decomposer is tested by comparing its output to that of Matlab. The invalid test cases include matrices that do not allow convergence within 30 iterations.

4.5 PSO UNIT TESTS

The Particle Swarm Optimizer requires a number of particles. The Particle Creator creates these particles. The particle creator test ensures that the particles created are within the bounds specified and the parameters are set as expected.

5. INTEGRATION TESTING

Integration testing is first done per component and then for the entire system.

5.1 COMPONENT BASED INTEGRATION TESTS

The component-based integration tests make use of mock objects for the other components that are not being tested.

- The Model Constructor, Model Center Calculator and Rotation Matrix Calculator as tested as a component. The Model Constructor is provided with the input data for all three cases described in 4.1.3. The Model Constructor then makes use of its helper classes (as opposed to mock objects) and the resulting transformation matrix is then compared to the expected transformation matrix. All the test cases that are invalid for one of the subcomponents are tested in this test case as well.
- The Lobby, Model Constructor and Model are also tested as a component. Tests are written to ensure that any new blocks introduced by the Model Constructor are added to the model's data without interfering with any current data in the model. Also a test should be written to ensure that an updated block correctly updates data in the model associated with the updated block and also does not alter any other data in the model.
  Finally a test will be written that when a smart block is classified as removed by the Model Constructor, the removal of the block is correctly propagated to the Model's data via the Lobby and that only the data associated with the removed block is removed.
- Database and Session Manager are tested as a component. The test involves the Session Manager requesting data from the Database and then comparing that to what is known to be in the database. The Database is also tested by asking for data that is not in the database

5.2 SYSTEM-BASED INTEGRATION TESTS

Since the Sketchup Blocks system makes use of real-time input, the only way that integration testing can be done effectively with reproducible results is by capturing data of various real-world inputs and feeding that information to the first step in the pipeline: the Interpreter.

The physical configuration of the cameras and blocks will be measured carefully. The digital data provided by ReacTIVision will be recorded and fed into the first stage of the pipeline. The calculated block and camera configuration can then be compared to the actual measured configuration to provide an estimation of the system's accuracy.

Since both the digital and physical data was recorded, the test can be performed repeatedly with the recorded input without additional environmental variables affecting the test results.

6. NON-FUNCTIONAL TESTING

6.1 THROUGHPUT

The number of camera events that can be processed is important since it determines the responsiveness of the system. A number of real world inputs will be recorded and fed into the system at different rates. Not all events require processing so a large number of inputs should be given to get a good estimate of the throughput of the system. If the system remains sequential the time before and after the events are given to the system can be recorded and used as an indicator of performance.

Once the system is concurrent both the time it takes to give input to the system and the time taken to process the input should measured. The time it takes to give the input to the system is important for the systems providing input to the model creator as they should not be held up.

The time it takes to process the information will determine how long it will take before the user sees change and is important for the user of the system.

The classes that perform intensive calculations are also timed by means of timestamps and various inputs. This data is also used as performance measure as well as to determine the most effective calculation methods (particular in the case of iteration methods).

6.2 RESILIENCE/RELIABILITY

The accuracy of the representation of the world is important. A part of this is tested during the calibration tests. The calculation of the world blocks depends heavily on the accuracy of the camera positions since block positions are calculated using information relative to the camera position.

With bad lighting ReacTIVision performs poorly, however this is outside of the project scope and cannot be compensated for by the system. If required, tests could be performed to measure system error in different lighting conditions.

7. COMMAND BLOCK BASED USER INTERFACE

Command blocks are used to interact with the system. This enables the user to interact with the system without the use of a keyboard or mouse. Tests will be written to ensure that a given command block does what is expected by testing each of the command blocks.

It should also be checked that no accidental actions are triggered and that the 3D view rotates around the origin with control. This will mainly be done manually by running the system for a series of stretches and observing its behavior.

8. TEST COVERAGE

The discussion of the unit tests has been divided in such a way that the unit testing of the entire functional project has been discussed. The unit tests will provide full coverage for all the functions in their service contracts - at the time of writing, not all of the unit tests have been implemented.

Test Coverage of Use Cases

- **New Project**
  The full system integration test as well as all the unit tests that test for correct initialization cover this use case.
- **Move Block**
  This is currently the most complex use case as it involves all the most complex units of the Sketchup Blocks system. The unit tests of all the pipeline components (section 4.1), the PSO and the math library (section 4.4) will play a part in covering this use case. All three component-based integration tests will also help to provide coverage. The full system integration test will also be vital to ensuring 100% test coverage for this use case.
- **Export Model**
  This use case is covered by the unit tests of the Collada loader and exporter. It is fully covered.
- **Process Calibration Block** is covered by means of unit tests (section 4.2) and will also be tested in the full system integration test.

All the testable functional requirements are covered by the non-functional tests.

9. GLOSSARY

Collada	Interchange file format for interactive 3D applications that is supported by Google Sketchup.
Command block	A smart block used to control the system (as opposed to model construction).
Construction floor	The flat surface on which the physical model is built and around which the web cameras are erected.
Fiducial markers	Small markers that will be attached to the physical blocks and used to identify them.
Google Sketchup	3D modelling program.
IoT	Internet of Things
MVC	Model-View-Controller, an architectural pattern
Object file	The Collada file that contains all the data of a particular model
PSO	Particle Swarm Optimization
ReacTIVision	Open source, cross-platform computer vision framework for tracking fiducial markers that are attached to physical objects.
Smart block	A physical block with an attached fiducial marker.
TUIO	A protocol that allows the encoding and transmission of a tangible interface component abstraction, such as tokens, pointers and geometries. This is the protocol by which ReacTIVision will communicate with the Sketchup Blocks system.