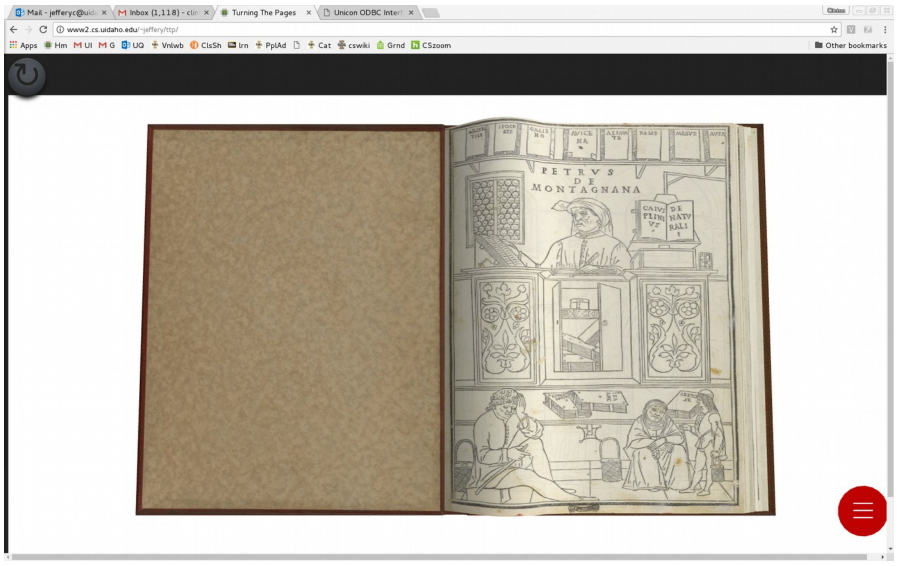
Design and Implementation of a Next Generation Turning the Pages Web Application

Clinton Jeffery and Michael Chung
June 28, 2017



Abstract

This report describes a new implementation of Turning The Pages for the world-wide web, developed on top of HTML5, JavaScript and WebGL. The new implementation runs well on many web browsers and platforms ranging from smart phones to high resolution desktop displays.

1. Introduction

The first NLM Turning the Pages Online web application (https://ceb.nlm.nih.gov/proj/ttp/) was developed in Adobe Flash approximately fifteen years ago. Due to security problems and loss of vendor support, the Flash platform is gradually disappearing as a web standard, having been subsumed by HTML5. In addition, the Flash Turning the Pages web application was designed for 1024x768 screens and does not scale to take advantage of the larger displays that are now standard. The combined changes in web standards and hardware capabilities means that today's Turning the Pages Online will not run on a good percentage of machines, and looks funny on most machines where it still can run.

A reimplementation of Turning the Pages for the web based on HTML5 had been contemplated well before the project described in this report, but it was unclear on what technology stack to proceed. While the Flash version essentially pre-cooked the 3D animation and provided it in the form of a movie, ubiquitous hardware support for 3D has created many alternatives. A fundamental underlying technology, WebGL, was developed that provides a full 3D API (OpenGL ES 2.0) to most modern JavaScript-enabled browsers. Atop the relatively low-level building block of WebGL, various JavaScript-based frameworks have been developed, such as three.js (https://threejs.org). However, JavaScript has some drawbacks. It was not designed for software engineering of large or complex software systems, and WebGL and OpenGL are focused on 3D output but do little or nothing in terms of user input. Also, JavaScript is a web-only language. An ideal implementation would provide the kind of cross-platform portability that allows one set of code to run on both the HTML5/JavaScript web and as a native application on PC desktops, and mobile operating systems such as Android and iOS.

The research question for this project is: can a game development framework called libGDX be used to meet the requirements of the next generation Turning the Pages web application? LibGDX applications are written in a mainstream programming programming language, Java. If the project were successful, it would mean that a new version of Turning the Pages Online can be developed that will be maintainable and extendable by Java developers using conventional software engineering processes. In this document, the project is referred to as TTP2, which is short for "Turning the Pages 2.0".

2. Requirements

The requirements for TTP2 were derived from the functional characteristics of the existing Turning the Pages Online application, the newer and more sophisticated Turning the Pages iOS application used on iPad and iPhone devices, and the capabilities of modern HTML5-based platforms. The functional requirements include:

- display of Maya animations: books' pages being turned
- display of text curations and hot spot notes
- play of audio from MP3 files
- zooming and panning the view on both desktop computers and touch-based devices
- provide navigation via thumbnail images of all pages

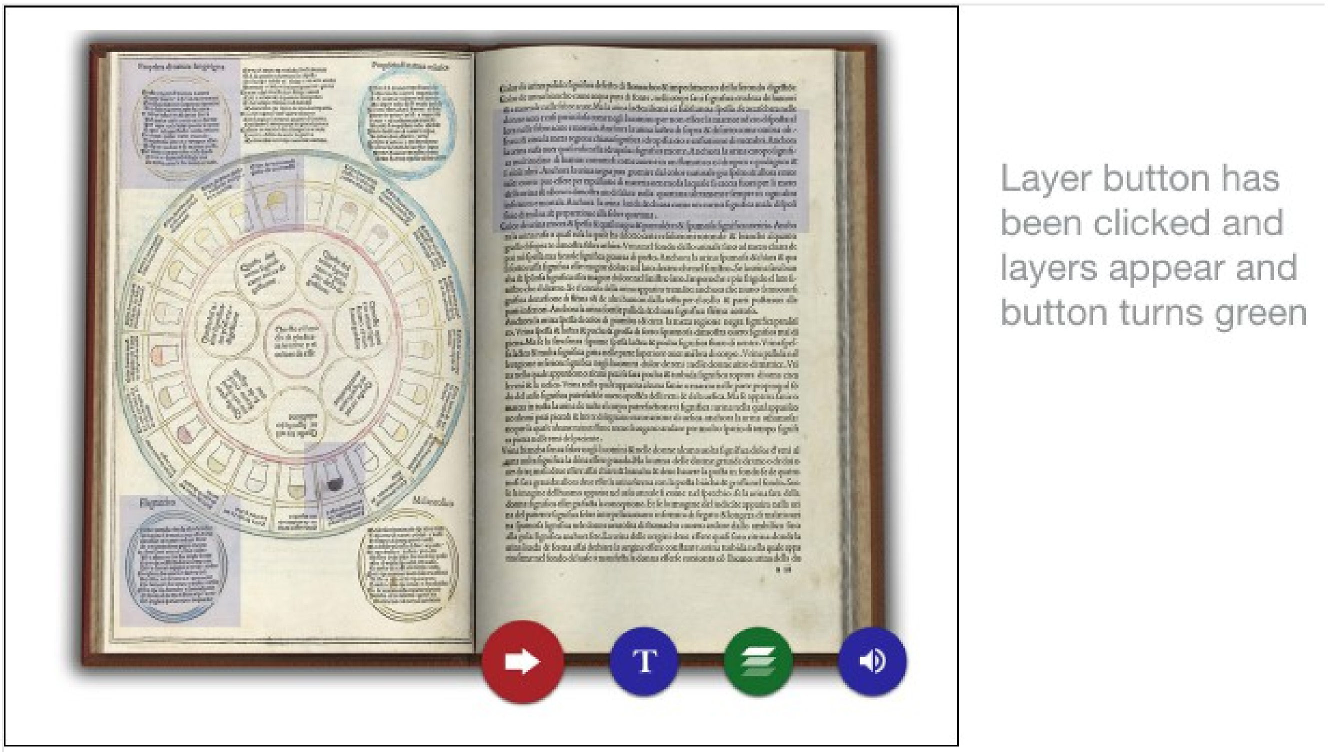
The non-functional requirements for the project include:

- execution within a wide range of modern web browsers that run HTML5

- ability to make good use of modern, high resolution displays
- download over the internet in a reasonable time on typical connection speeds

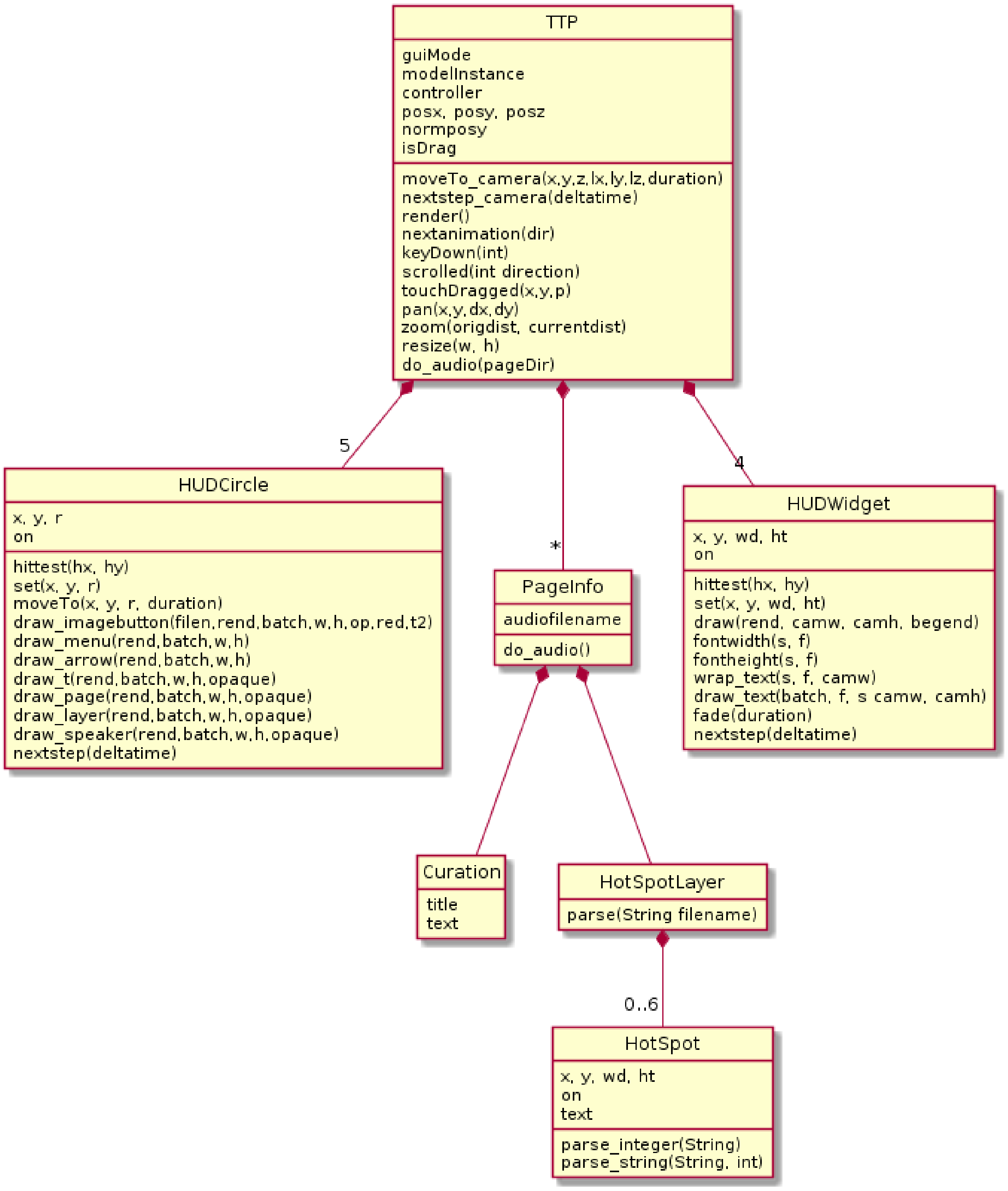
3. Design

The project design of TTP2 consists of a graphic and user interface design, plus a software design. The graphic user interface design was prototyped by Michael Chung with a series of four sample screenshots such as the following.


Layer button has been clicked and layers appear and button turns green
T

The vast majority of TTP2's software design is dictated by the libGDX framework. A crude summary is presented here. LibGDX divides the application into a portable multi-platform *core*, and for each operating system, a platform-specific *launcher*. Almost all the code is portable and located in the core.

The TTP2 core revolves around a primary application class named TTP. The TTP class is a singleton; it contains references to all other class instances. TTP.java contains more than half the application source, including code to control the 3D turning-pages Maya model. It implements libGDX user input processing interfaces, both the standard InputProcessor for keyboard and mouse, and the GestureDetector.GestureListener interface for touch input gestures such as panning and zooming. The rest of the classes in the TTP2 application are the objects that the TTP object contains, primarily consisting of the few simple components that comprise TTP2 user interface, or "heads up display" (HUD).



In this diagram, a HUDCircle is a circular button that may be drawn in red, blue, or green with one of six face images. A PageInfo object contains the annotations information associated with a given page; the TTP object contains a HashMap named pagemap that maps (two) pageDirs to the corresponding PageInfo object. A Curation is a simple object holding the title and text read in for one page.

A HotSpotLayer is an object holding the hotspots for one page. A HotSpotLayer knows how to parse one of the supplied XML files containing this information. The HotSpotLayer object contains its HotSpot objects in a LinkedList.

Lastly, a HUDWidget is a simple GUI component capable of rendering a text in a rectangle. The HUDWidget is overloaded slightly, serving for both the Curation text, the HotSpot texts, the background of the thumbnails (no text, images instead), and the small zoom indicator that fades out after showing a zoom change.

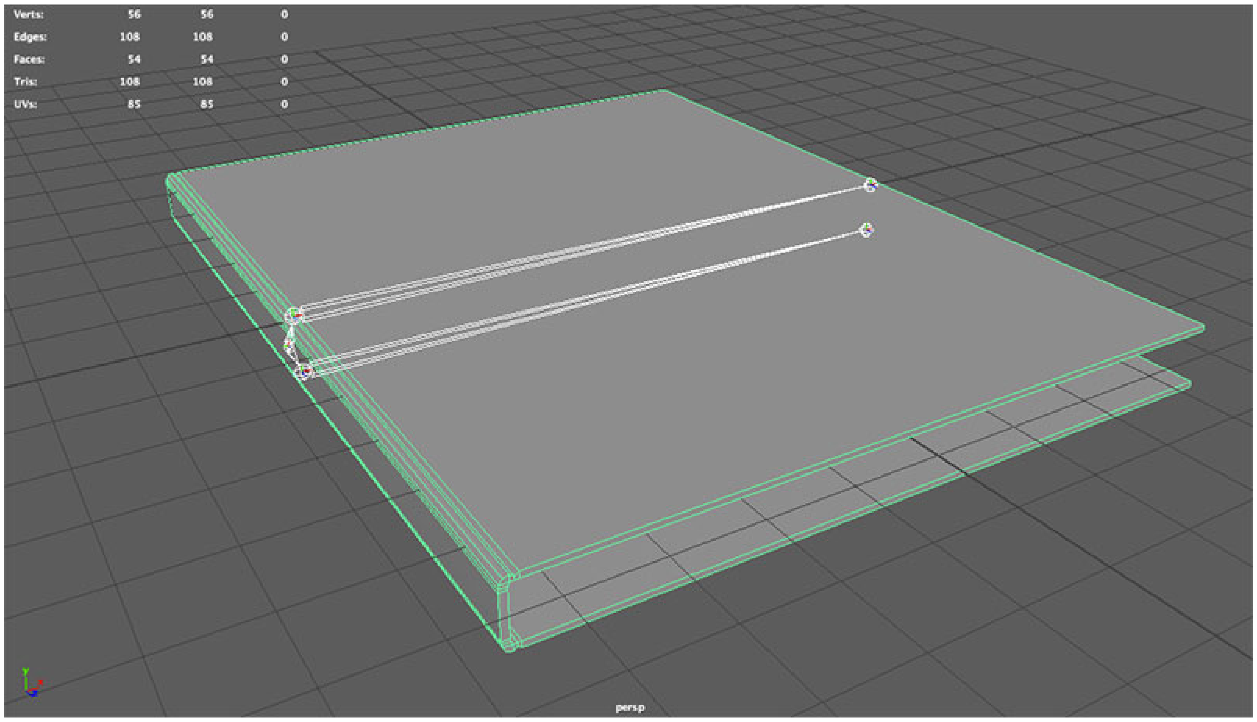
The common design elements shared by HUDCircle and HUDWidget include: normalized coordinates such as x and y in the range [0,0] to [1,1], almost everything being public, objects knowing how to draw themselves various ways, and crude linear change support for visual effects.

4. Implementation

The implementation of TTP2 includes both 3D modeling and animation in Maya, and Java application development in libGDX. It is at the intersection of these two efforts that the most interesting implementation challenges were faced and overcome.

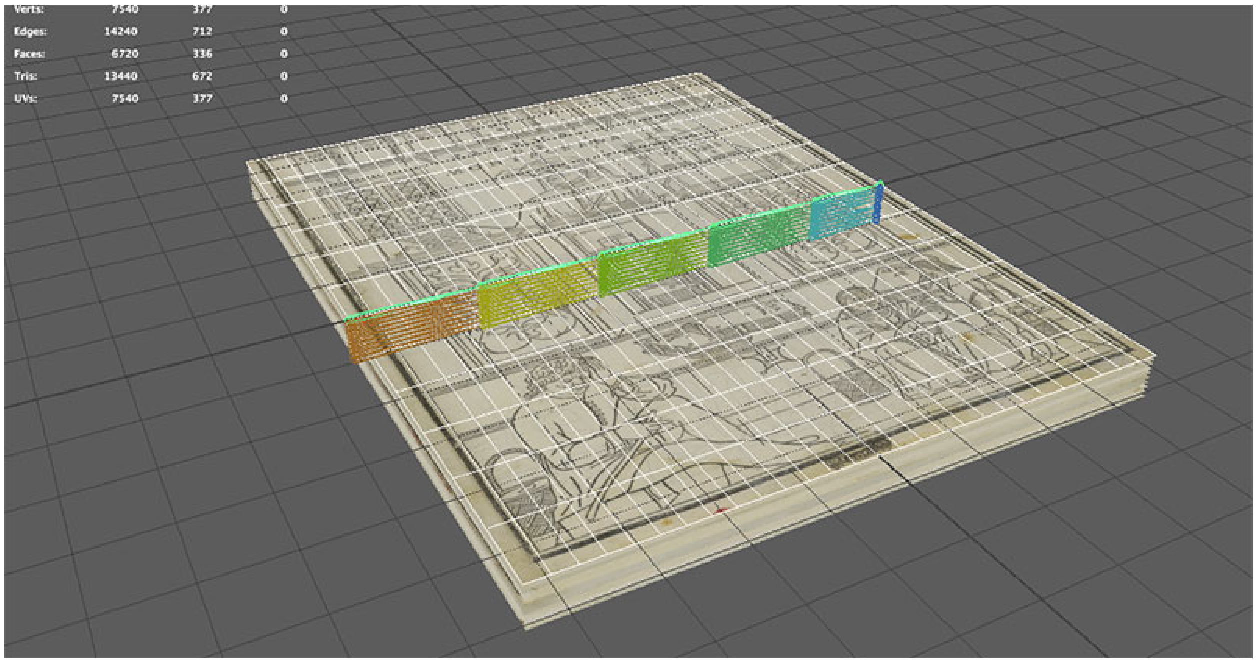
4.1 Development of LibGDX-Compatible 3D Models in Maya

Due to limitations in the FBX format, libGDX animations such as the book page turns use joints (bones) rather than other animation techniques used previously in Turning the Pages. The cover is a poly cube with faces extruded to form a closed book. A central joint is placed in the middle of the spine with 2 joints for each part of the cover (2 for the front cover and 2 for back)


```
Verts:   56   56   0
Edges:   108  108  0
Faces:   54   54   0
Tris:    108  108  0
UVs:     85   85   0
```

The book is bound to the joints using a soft binding and weights are painted in Maya so the joints correctly affect the underlying polygon model.

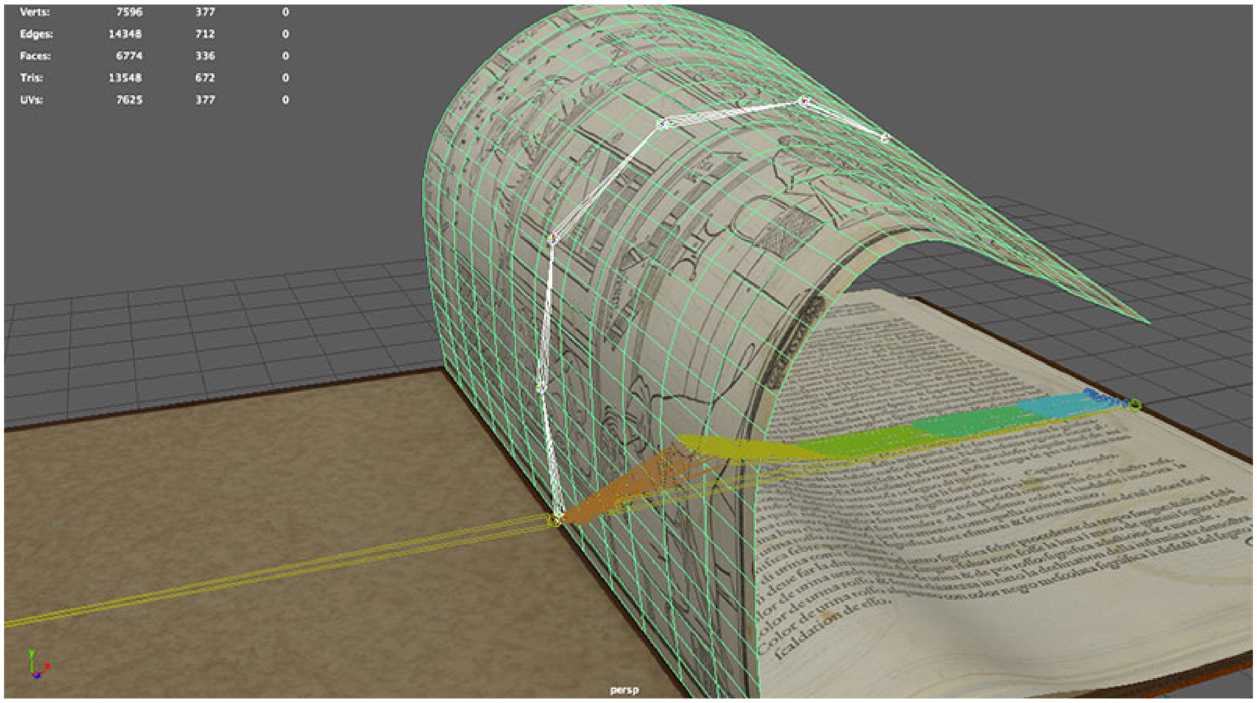
Each page is a polygon plane bound to a series of 5 joints.



```
Verts:     7540    377    0
Edges:    14240    712    0
Faces:     6720    336    0
Tris:     13440    672    0
UVs:       7540    377    0
```

The plane is again smooth bound to its corresponding joints with the weights painted in Maya to give a smooth curvature when animated. Each joint's weights are flood smoothed for greater smoothness (essentially blurring the painted weights). The plane is 1-sided (backface culling turned off) and then duplicated and flipped. The weights from the page are copied and pasted on to the duplicated and reversed page. This allows for the front and back of each page to be displayed as separate objects but animated by the same base joint system. Two separate objects were created for front and back so they can be textured separately. This process was repeated for all of the pages (10 in total).

Once all of the models are built and bound to joint systems, the cover opening and each page turn were animated in Maya by hand. The animation is handled by rotating the joints. So a page turn is simply a rotation of the parent joint 180 degrees. The child joints are then animated incrementally to simulate page turning.



Verts:	7596	377	0
Edges:	14348	712	0
Faces:	6774	336	0
Tris:	13548	672	0
UVs:	7625	377	0

The Maya model is exported to FBX format using the game exporter option in Maya. As all of the aforementioned animation is handled in the timeline, animation clips are named based on their timecode. For example, the open-book animation clip starts at frame 1 and ends at frame 15. Page01 is frame 20 – 35. All page turn animations are 15 frames.

In Maya, under settings, set the Up axis to Y and check the bake animation checkbox. Export in Binary to the FilmBox FBX 2014/2015 format. For the most part, the exporter is left on default settings. This exports to a single .FBX file suitable for input to LibGDX's fbx-conv program.

4.2 TTP2 as a Java LibGDX Application

Although the project runs in web browsers as an HTML5, JavaScript, and WebGL, it is written in about 2,200 lines of Java using the libGDX library. For its HTML target, LibGDX takes the Java code and compiles it down to JavaScript using the Google Web Toolkit (GWT). This link is the most fragile part of the whole project, and the reason why it was unknown whether the project would be practical and feasible. GWT is not a full implementation of Java, it supports a subset of the language and its libraries. LibGDX supports many output targets, and HTML5 is its most fragile; many libGDX programs that run well on the desktop or as Android applications do not run in HTML5.

The implementation of TTP2 was successful, but during the course of development bugs were encountered that necessitated patching and building from a custom copy of libGDX. In particular, it took numerous attempts before a Maya animation was produced that would run in libGDX, and when it finally did, it would not run in HTML5 on the current version (1.9.6 or 1.9.7-snapshot). It was discovered that the Maya animation would run in the previous version (1.9.5) of libGDX that is the default when creating new libGDX projects. In turn, that version of libGDX would not display text in HTML5 due to a font loading bug which had since been resolved. To get text display and Maya animations together in the same application required applying a bugfix to the 1.9.5 source code and rebuilding libGDX. Similarly, code to display the button faces' PNG images ran on the desktop, but the code to display them had to be modified to run in HTML5.

Besides all this debugging and versioning, the majority of the implementation of TTP2 consisted of the following tasks:

1. playing of media: the Maya animation and audio MP3 files
2. superposition of a heads-up display (HUD) of 2D opaque and translucent widgets atop the 3D scene showing the book.
3. parsing Turning the Pages curation and hotspot files
4. processing many kinds of user input, including keystrokes, mouse clicks and drags, and touch gestures.
5. handling varying screen resolutions and processing of resize events

Of these, Java and libGDX were very good at (1) and (2), mediocre at string processing needed for (3), very good at (4), and fairly good at (5). This is largely proportional to the degree to which the task occurs in the videogame domain from which libGDX originates.

On the user interface (HUD) components, motions, fades and such are handled via a second set of “target” attribute values and a method nextstep() called each frame to track time changes and interpolate new coordinate attribute values.

4.3 Preparing the Sources for Building TTP2

The TTP2 source distribution is delivered as a .zip file that unpacks with two subdirectories, named ttp/ and libgdx-1.9.5/. TTP2 is a libGDX Java application. Developers should install Java; building minimally requires that java and javac commands are on the path; probably Maven (mvn) is also needed, and possibly also JAVA_HOME and/or other Java-related environment variables. The TTP2 libGDX is modified slightly from stock libGDX 1.9.5. LibGDX can be learned by reading books, but its developer culture seems to revolve around watching youtube videos. Such videos may be helpful in learning how to modify or extend TTP2, or to import TTP2 into an IDE.

The TTP2 source files include a top-level file, build.gradle. When configured correctly, compared with the generic version constructed by the libGDX setup program, this file will contain a change to use Lightweight Java Gaming Library 3, which is routine for libGDX programs, and three lines that point gradle to the custom modified version of libgdx used for this project. A new developer on this project will have to adjust the paths used, to indicate where libGDX has been unzipp’ed instead of /home/jefferycl/ as shown below.

```
project(":desktop") {
   apply plugin: "java"

   dependencies {
      compile project(":core")
      compile "com.badlogicgames.gdx:gdx-backend-lwjgl3:$gdxVersion"
      compile "com.badlogicgames.gdx:gdx-platform:$gdxVersion:natives-desktop"
      compile "com.badlogicgames.gdx:gdx-box2d-platform:$gdxVersion:natives-desktop"
   }
}
```

```
project(":html") {
    apply plugin: "gwt"
    apply plugin: "war"

    dependencies {
        compile project(":core")
        compile files('/home/jefferycl/libgdx-1.9.5/dist/gdx-backend-gwt.jar')
        compile files('/home/jefferycl/libgdx-1.9.5/dist/sources/gdx-sources.jar')
        compile "com.badlogicgames.gdx:gdx-backend-gwt:$gdxVersion:sources"
        compile "com.badlogicgames.gdx:gdx-box2d:$gdxVersion:sources"
        compile "com.badlogicgames.gdx:gdx-box2d-gwt:$gdxVersion:sources"
    }
}

project(":core") {
    apply plugin: "java"

    dependencies {
        compile files('/home/jefferycl/libgdx-1.9.5/dist/gdx.jar')
        compile "com.badlogicgames.gdx:gdx-box2d:$gdxVersion"
    }
}
```

4.4 Compiling and Running TTP2

This section will briefly discuss the basics of compiling and running TTP2, once the sources have been setup properly. Both of the two current build targets discussed below are executed from the top-level ttp/ directory.

TTP2 is compiled and executed locally as a desktop application on Linux using the command

```
./gradlew desktop:run
```

If a UNIX make(1) program is on your system, a makefile is provided, so on Linux one can just say make to run this command. TTP2 has not been built yet on Windows, but given the proper installed Java, libGDX's gradlew.bat file allows you to just say gradlew desktop:run in the same way.

TTP2 is compiled as a web application using the command

```
./gradlew html:dist
```

This generates a complete webpage with subdirectories containing compiled javascript and binary assets in html/build/dist. The command make htm will run this command. Coffee break caveat: on my very fast NLM machine, this make takes around 75 seconds.

Changing down into the html/build/dist subdirectory, one can copy the entire hierarchy out and into a web server. For example, scp -r * myname@destination:location/ttp copies out each new build into a ttp/ directory on a remote web server. On the web server machine, the main requirement is that the whole TTP application contents are readable and the directories are all executable. On a remote Linux web server, chmod -R ugo+rx ttp is easy and works, but might place execute permissions on more files than is desirable for a production site. Typically a developer should then clear their browsing data (for example, under settings...More tools in Chrome) prior to refreshing or visiting the page, in order to make sure current code is being used.

5. Testing

As a research project, TTP2 has not yet been put through rigorous software testing. The primary testing conducted thus far is portability testing. A basic question was whether the developed software would run on a wider range of devices than the current Turning the Pages Online web application.

Windows Chrome	yes
Windows IE	yes
Windows Edge	yes
Linux Firefox ESR 52.1.0	**no**, black screen
Linux Chrome	yes
iOS (iPad) Safari	**no**, black screen
OSX Safari 10.0	yes
Android 7.0 Chrome	yes, portrait mode issues
Android 7.0 Firefox 51.0.4	yes, portrait mode issues
Android 6.0.1	yes, portrait mode issues
Android 5.1.1 Galaxy Tab A6 SM-T280, 2016	**no**, flashes until it goes black. Best known clue: • http://badlogicgames.com/forum/viewtopic.php?t=16157&p=69444 claims it has to do with setScreen()
Android 5.0.1	yes, portrait mode issues

Additional testing that would be valuable to the HTML5 web implementation includes: unit testing of the implemented classes, as well as testing on a wider range of web browsers and operating systems. A major issue is resizing: resizing the window does not necessarily translate into a web browser changing the virtual canvas it allocates to an open web page. It has scrollbars, after all.

6. Future Directions

The obvious next step with this project is to develop the new TTP2 versions of all fifteen or so existing Turning the Pages titles. For the most part, the TTP2 code is setup to accommodate this task easily. In addition, there are several potential opportunities to make the Turning the Pages system more widely available, such as a native Android version, using the TTP2 LibGDX implementation.

6.1 Adding New Turning the Pages Titles

TTP2 was written initially for de Ketham's Fasiculo de Medicina. This section describes the tasks required to adapting it for other Turning the Pages books. In a perfect implementation, this would involve no Java code changes, only a change of data files. However, every Turning the Pages book seems to include its own special features. The more new, custom behavior that is required by a new title, the more custom Java code will have to be written. This section can only anticipate the common elements of any new title.

1. Create an assets subdirectory, under android/assets, named to distinguish the new title from the others. Due to a quirk of the libGDX build process and to allow for a future Android port without file reorganization, all target platforms place their assets under the android/assets directory, not just android builds. Write the name of your assets subdirectory as the first line of assets/ttp.conf. For example: ketham in ttp.conf implies books-specific assets are to be found in android/assets/ketham/.

2. Create a .G3DJ model file corresponding to the Maya animation file, exported from Maya in FilmBox FBX format. At present, the model file is required to be in ASCII format (G3DJ, not G3DB) due to step 3 below. Converting FBX to G3DJ format is accomplished using a tool called fbx-conv. On Linux I used a file fbx-conv-lin64 and had to export an LD_LIBRARY_PATH with a directory containing a libfbxsdk.so. TTP2 reads the model filename from the second line of of ttp.conf, for example: ketham-fix.g3dj. Create texture images as needed/used in the model file. Texture images' dimensions must be powers-of-two, such as 1024x2048. There is some question as to what image formats libGDX supports on all platforms. JPEG files work.

3. Establish a mapping from book "pages" to animation names. The model file contains a set of animations corresponding to page turns. Since G3DJ is an ASCII text format, TTP2 extracts a String array named animation_names from the "id" fields of the entries in the animations section of the G3DJ file. The animation names are taken to be sequential, starting from the front, going through all the forward page turns in sequence to the end. The backward page turns are performed by playing animations in reverse. The mapping for any particular page turn requires both the page and the direction of the turn. TTP2 uses an integer, pageDir to encode both the page and the direction, with numbers greater than pages/2 working backwards from the end. (Example: forward numbers on a 12 page book are 0..11. pageDir 12 turns you back to where you were on 10, 13 turns you back to where you were on 9, etc.

4. Establish a mapping from book "pages" to PageInfo objects containing text curations and hotspot data. "Pages" is in quotes because, once again, the pageDir variable has twice the actual number of pages to track going forward and backward. The mapping from pageDir to PageInfo objects is encoded in the pageinfo section of ttp.conf and implemented by a HashMap named pagemap in TTP.java.

5. Add any new user interface components or behavior required by the title. This is open-ended and requires Java proficiency and libGDX and OpenGL skills.

6.2 Future Directions with the Java and libGDX Implementation

LibGDX has several targets besides HTML5. From the same Java source code, with perhaps minor customization, native Android and iOS clients can be generated. Although iOS is already well-served by the existing iOS Turning the Pages application, there are 4-5 times as many Android devices deployed as there are iOS devices. Although the application as developed runs in a Web browser on most Android devices, the libGDX Android application target should be implemented in preliminary form and evaluated to see if it has better usability, or runs on a wider set of Android devices, than the HTML5 implementation.

Numerous aesthetic improvements are desirable, ranging from improved shadowing/lighting in the page turns, to partial page turns and a more realistic physical model.

6.3 Lessons Learned

While libGDX proved capable of meeting the requirements of the TTP2 project, it is not flawless. Prior to this project, students had reported that it didn’t support its HTML5 target adequately and had avoided writing games for the HTML5 target. The TTP2 project confirms that the HTML5/JavaScript target is fragile and has bugs.

The Maya animations for Turning the Pages did not work out of the box in libGDX. It required several iterations of experiments to find a combination of options to Maya that would allow it to export animations that libGDX can display.

Even after the Maya animations were displaying well in libGDX using its regular desktop application format, the TTP2 Maya animation would not run on HTML5. The discovery that they ran on the previous version of libGDX that is its default, and that the bug was specific to the current version of libGDX, was a happy accident. On the other hand, the previous version of the HTML5 platform had a font loading bug that had been fixed in the current version. The short term solution was to apply the font fix to the source code of the previous version of libGDX and rebuild it. For a longer term solution, it would help if we can make the TTP2 code available to libGDX maintainers, and ask them to use it to debug the current version of libGDX.

7. Conclusions

An HTML5/JavaScript implementation of Turning the Pages has been developed successfully using Java and LibGDX. The research was performed on the de Ketham book, and other Turning the Pages books would require some additional coding for alternative user interface elements such as the scroll animation. Nevertheless, the research implementation meets core requirements and since it is written in Java, it might be possible to find a local Java programmer willing to work on additional titles.

References

libGDX Cross-platform Game Development. https://libgdx.badlogicgames.com/
Michael Chung. Libgdx-UI. https://xd.adobe.com/view/5f22a63e-1459-490c-9260-8359b2df29dd/
Google Web Toolkit, http://www.gwtproject.org/.

Appendix A: the ttp.conf File

The following ttp.conf file, for the Ketham title, illustrates how title-specific information is encoded in the assets/ directory so that it doesn't have to be hardwired into the Java source code. The configuration file must be named ttp.conf in the android/assets directory, and it is at present very primitive. The first line has to be an assets/ subdirectory to use, the second line a model file, then the curation filename, the pageinfo section, and the thumbnails. The code to read this format does not yet include a real parser that would provide flexibility.

```
assets ketham
model ketham-fix.g3dj
curation TextBook9.txt
pageinfo (curation,hotspots,audio) {
0,22 book9_text2,HotSpots9_1.plist,Book9_2_audio.mp3
1,21 book9_text3,HotSpots9_2.plist,Book9_3_audio.mp3
2,20 book9_text4,HotSpots9_3.plist,Book9_4_audio.mp3
3,19 book9_text5,HotSpots9_4.plist,Book9_5_audio.mp3
4,18 book9_text6,HotSpots9_5.plist,Book9_6_audio.mp3
5,17 book9_text7,HotSpots9_6.plist,Book9_7_audio.mp3
6,16 book9_text8,HotSpots9_7.plist,Book9_8_audio.mp3
7,15 book9_text9,HotSpots9_8.plist,Book9_9_audio.mp3
8,14 book9_text10,HotSpots9_9.plist,Book9_10_audio.mp3
9,13 book9_text11,HotSpots9_10.plist,Book9_11_audio.mp3
11,23 book9_text0,,Book9_0_audio.mp3
}
thumbnails {
spread00.png
spread01.png
spread02.png
spread03.png
spread04.png
spread05.png
spread06.png
spread07.png
spread08.png
spread09.png
spread10.png
spread11.png
}
```

Appendix B: PlantUML source

The PlantUML (plantuml.com) source for the UML class diagram depicting TTP2's design is presented here so that it can be preserved and edited in future revisions of the document.

```
@startuml
hide circle
hide empty methods
hide empty fields
TTP *-- "5" HUDCircle
TTP : guiMode
TTP : modelInstance
TTP : controller
TTP : posx, posy, posz
TTP : normposy
TTP : isDrag
TTP : moveTo_camera(x,y,z,lx,ly,lz,duration)
TTP : nextstep_camera(deltatime)
TTP : render()
TTP : nextanimation(dir)
TTP : keyDown(int)
TTP : scrolled(int direction)
TTP : touchDragged(x,y,p)
TTP : pan(x,y,dx,dy)
TTP : zoom(origdist, currentdist)
TTP : resize(w, h)
TTP : do_audio(foo)
HUDCircle : x, y, r
HUDCircle : on
HUDCircle : hittest(hx, hy)
HUDCircle : set(x, y, r)
HUDCircle : moveTo(x, y, r, duration)
HUDCircle : draw_imagebutton(filen,rend,batch,w,h,op,red,t2)
HUDCircle : draw_menu(rend,batch,w,h)
HUDCircle : draw_arrow(rend,batch,w,h)
HUDCircle : draw_t(rend,batch,w,h,opaque)
HUDCircle : draw_page(rend,batch,w,h,opaque)
HUDCircle : draw_layer(rend,batch,w,h,opaque)
HUDCircle : draw_speaker(rend,batch,w,h,opaque)
HUDCircle : nextstep(deltatime)
TTP *-- "*" PageInfo
PageInfo *-- Curation
PageInfo *-- HotSpotLayer
PageInfo : audiofilename
Curation : title
Curation : text
HotSpotLayer *-- "0..6" HotSpot
HotSpotLayer : parse(String filename)
HotSpot : x, y, wd, ht
HotSpot : on
HotSpot : text
HotSpot : parse_integer(String)
HotSpot : parse_string(String, int)
TTP *-- "4" HUDWidget
HUDWidget : x, y, wd, ht
HUDWidget : on
HUDWidget : hittest(hx, hy)
HUDWidget : set(x, y, wd, ht)
HUDWidget : draw(rend, camw, camh, begend)
HUDWidget : fontwidth(s, f)
HUDWidget : fontheight(s, f)
HUDWidget : wrap_text(s, f, camw)
HUDWidget : draw_text(batch, f, s camw, camh)
HUDWidget : fade(duration)
HUDWidget : nextstep(deltatime)
@enduml
```

Appendix C: Modifications to LibGDX 1.9.5

The specific changes made to libGDX 1.9.5 are presented here for the sake of completeness; they should already be applied if you are installing from the TTP2.zip distribution. The changed file is named /libgdx-1.9.5/gdx/src/com/badlogic/gdx/graphics/g2d/BitmapFont.java. The changes made to it consist of about 6 lines which fix its ability to load fonts. These changes were made as libGDX moved to version 1.9.6, but version 1.9.6 and newer versions contain a bug which prevents TTP2's page turning animation from displaying properly on the HTML5/JavaScript target, a core requirement for our implementation.

```
$ diff BitmapFont.java BitmapFont.java~
532c532
<       if (matcher.find()) {
---
>       if (matcher.matches()) {
535,536c535,536
<                         int pageID = Integer.parseInt(id);
<                         if (pageID != p) throw new GdxRuntimeException("Page IDs must be
indices starting at 0: " + id);
---
>                         int pageID = Integer.parseInt(id.substring(3));
>                         if (pageID != p) throw new GdxRuntimeException("Page IDs must be
indices starting at 0: " + id.substring(3));
538c538
<                         throw new GdxRuntimeException("Invalid page id!!: " + id, ex);
---
>                         throw new GdxRuntimeException("Invalid page id: " + id.substring(3), ex);
542,543c542,543
<                         matcher = Pattern.compile(".*file=\"?([^\"]+)\"?").matcher(line);
<                         if (!matcher.find()) throw new GdxRuntimeException("Missing: file");
---
>                         matcher = Pattern.compile(".*file=\"?([^\"]*+)\"?").matcher(line);
>                         if (!matcher.matches()) throw new GdxRuntimeException("Missing: file");
```