

Computer Science Widening the STEM Education Spectrum

Christopher J. Morack
Department of Computer Science
Tennessee Technological University
Cookeville, TN, USA
cjmorack42@students.tntech.edu

William Eberle
Department of Computer Science
Tennessee Technological University
Cookeville, TN, USA
weberle@tntech.edu

Abstract— Science, Technology, Engineering and Mathematics (STEM) education is slowly becoming an important part of American culture. STEM educators try to promote ‘hands on’ science, where students can actually interact with and see the results of their work. Unfortunately, not all paths of education for STEM related fields can have exciting, interactive teaching methods. At the Millard Oakley STEM Center, we have taken advantage of the interactive experience in our planetarium show. In this paper, we take an in-depth look at the Definiti Theater System and the software that runs it, Digital Sky 2. This software package is created by Sky-Skan and is considered to be the standard for many new planetariums across the country and beyond. We take the software in new directions by building new elements through the Sky-Skan scripting engine and by also exploring its 3D engine for creating a novel experience. The primary purpose of this paper is to provide a roadmap of observations and enhancements for other educators that wish to improve the learning experience of students and visitors to their planetarium.

Keywords—STEM, education, planetarium, Digital Sky 2, 3D

I. INTRODUCTION

The goal of STEM education centers is to engage students of all ages enough so that they will find an interest in a STEM career path. The scientists, engineers, and mathematicians of tomorrow are hidden among the children of today. Any moment of a child’s life could be that special moment that shapes who they will become. To try to capture the minds and interests of students, STEM programs employ a hands-on technique to their offered programs [1]. This approach sets a STEM program apart from other programs. Instead of students simply learning about the laws of physics, they will learn about them as well as build a rocket with which they experience physics in action. They can then compare and talk about their results, and what they observed and experienced.

It is in this area where programs like astronomy shows tend to fail. As the very name implies, it is a “show”. Regardless of how knowledgeable or enthusiastic the program host may deliver the show, it is still the same material. When a student experiences the show, they know their peers will see the exact same thing, outside of the differing questions other students may pose. Few students will talk about their experiences in an astronomy theater with their peers because they all have the exact same experience.

In our previous shows, we used minimal 3D assets and relied heavily upon the speaker to convey what various dots and circles and lines on the screen meant. In shows that did feature 3D models, none of the models were original or custom built to showcase that particular show. The shows were built only around what was pre-existing within the system, and limited the creativity of the developers. Also because a show was ‘a show’, the thought never occurred to the developers to allow the on-screen action to be directed by the students.

To better achieve the expected STEM experience from our existing astronomy program, a redesign of the previous creation methods was necessary. What we desired was a new program that could support the hands-on approach to learning through a virtual experience. Ideally, the program could involve numerous choices, each with different events and moments. In order to harness their interest and spark conversation outside of the show between students, events needed to be different and interesting – or at least different enough that each student could have his or her own story. Key events would tie back together, so the goal of the educational material would still be achieved, but additional material would be different from student to student.

This paper will discuss the topics associated with not only the discovery process and the angling of the concepts of a new interactive astronomy program, but also delve into the details of the program itself. The topics will include methods of setting up interactive features using Sky-Skan’s Definiti planetarium software, Digital Sky 2, to achieve a fully interactive environment [2]. Pitfalls, workarounds, and tips encountered by our team will be discussed at each step. While much of the paper will be covering the Digital Sky 2 software, the basic concepts used can cover a wide variety of software where an educator may be having a difficult time building interactive features for their students. The goal of this paper is to provide a roadmap for transforming a non-interactive presentation into something that will not only further STEM objectives, but also will capture the attention of students.

II. RELATED WORK

For our approach to be useful to others, we researched a myriad of ways to implement the necessary interactive features. However, up until now, little work has been done in this area. We were able to discover some of the simpler features through work by others by reading the private forums associated with Digital Sky 2 purchasers. To the best of our

knowledge, more complex functionality such as asset creation and implementation has never been fully explained or well documented.

Planetarium web pages across the United States prove they follow a standard approach to their astronomy centers. They tend to feature a scripted planetarium segment and a non-interactive, pre-recorded movie. Generally the show’s commentary is prerecorded, occasionally by a person of note within the science community or an author. Any interactive features available by the system are either unused, or used in such a way that there is no dedicated show built specifically around those capabilities.

One such example of this is The Morehead Planetarium and Science Center at the University of North Carolina at Chapel Hill [3]. The shows look spectacular, but the experience is the same for everybody. Since there is limited or no interaction with the audience, the experience is an educational movie.

While most planetariums follow the route of the non-interactive feature, the Lawrence Hall of Science’s planetarium at the University of California, Berkeley does provide an interactive experience [4]. They also use the Digital Sky 2 software – the same software that we use at the Millard Oakley STEM Center. In this case, the show is hosted by a real person and the host can answer questions posed to them and direct the screen to whatever object a person may want to see. This type of feature is fully within the operational boundaries of the Digital Sky 2 software, but it still does not fully realize the goals presented in this paper.

In short, to the best of our knowledge, what we are presenting in this paper is novel. For most planetariums the word interactive describes a show run by a live commentator. The commentator controls the system in a pre-existing way, such as to view a certain object. The commentator has control of the *navigation mode* already built into their software. This method is useful, but it will not grab the imagination of an audience and pull them into a different world where they are in the driver’s seat. In fact, this interactive method is traditionally how we have used the software in the past. This level of interaction lacks the excitement of what we are aiming to do with our interactive story approach to education. We believe that the Digital Sky 2 software has a great deal of ‘interactive potential’. So, our goal is to capture the essence of what ‘interaction’ means in an educational setting, and refocus it into something that will grab students’ attention by literally pulling them into the story.

Through a system such as Digital Sky 2, we have the capability to create a lasting, educational experience. However, the capability to do that is not simply inherent within the software in and of itself. There are certain techniques, which when applied, have helped to move our presentations to another level of excitement, where the student can literally take part in an adventure driven by education.

In other words, our goal is to not just show a movie. We want to surprise the audience with a story where *they* are the author of something exciting. The following is our proposed approach to implementing a truly interactive planetarium show,

with detailed steps and examples that will allow other practitioners to do the same.

III. PROPOSED SOLUTION

A. Storyboard / Event Flowchart

Before beginning to build a show, it is important to answer the following questions: What do you plan to teach? What do you plan to talk about? What do you want to start with and what do you want to end with? Even for practitioners who are not using the Digital Sky 2 software package, this step is critical to any feature event, whether it is interactive or non-interactive. It is easy to skip this step and dive directly into building a vision, but this haphazard approach will often result in a confused, incoherent project. To get around future difficulties, it is good practice to create a *storyboard* at the beginning of development. A storyboard is not a new concept, and is something many people familiar with the entertainment industry will recognize [5].

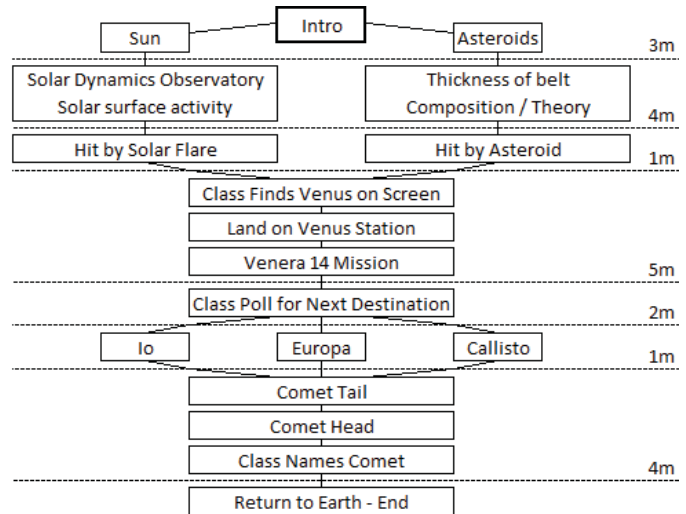


Figure 1: Example Storyboard Flowchart

To implement one’s strategy, one must storyboard ideas using a medium that is easy for all to view and edit freely. For our purposes, we storyboard our ideas on large white boards. Ideas undergo numerous changes until brain storming is finished. Our initial storyboard consists of a flowchart of topics we want to cover, and the amount of time we want to take talking about each topic. Lines connect the different branches the feature will take (Figure 1).

Since our feature will have key topics we want to cover, we use a ‘diverge and converge’ system. Each major topic will be talked about, followed by an ancillary topic the students would choose as a next destination. Those destinations would ultimately return to our main topic again. This initial flowchart storyboard not only gives us a way to ‘see’ how the show will play out, but it also gives us rough estimates for timing. These estimates can be taken into action for writing the speaking portions of the show.

Additional work with a storyboard may be necessary to achieve the desired look and feel. It may be necessary to draw

out individual scenes, complete with timing for smaller events, including the timing for audience participation. For each individual event in the flowchart, we follow the standard method of storyboarding. We draw comic book-like panels of pictures describing what occurs. This helps us further recognize assets we need to create, as well as what we would say and when.

With the storyboard visible to the development team, all creative thoughts and ideas are never lost. Since a 20-minute show may take a year to develop, preserving ideas for later implementation is a critical key to success. The storyboard also provides a convenient metric to measure the completion percentage of the overall presentation.

B. Understanding the Model & Brief History

Creating models is necessary for custom content, and understanding the format of the model file is necessary to create professional looking models. Digital Sky 2 uses DirectX models in the .x format [6]. This is an older and depreciated format, and to the best of our knowledge no fully compatible Digital Sky 2 model exporter exists. For certain features, such as applying *shaders*, additional model editing must be done.

We initially used Blender to export our models to the DirectX format [7]. In our version of Blender, we enabled the ability to export to the DirectX format through the *Add-ons* menu. However, we discovered some issues exporting certain materials, and creating animations was difficult as well. This led us to look for alternatives.

What we use now is Autodesk 3D Studio Max 2012 [8]. Autodesk 3D Studio Max is very expensive to buy, but for students it is free to use. Although we are not able to build DirectX *shaders* compatible with the Digital Sky 2 software using the *shader* creation tools within 3D Studio Max, we are able to resolve animation issues and ease of exportation. Sky Skan also uses 3D Studio Max to create their content, which further justifies our selection of this software. However, between the version Sky Skan uses and the current version of 3D Studio Max, the ability to natively export to DirectX models has been removed.

Therefore, since we did not want to lose the great modeling and animation features present in the newer versions of 3D Studio Max, we had to locate a third party exporter. After a lot of failed attempts, we found Padasoft's DirectX Exporter [9]. This exporter covers all of the bases, including animation, and even allows for direct export of models with a basic texture map material without the need to edit the .x file.

Digital Sky 2 can be very picky if certain modeling actions are taken. If an object is not modified at the *sub-object level*, it may be deformed in the Digital Sky 2 environment. Translating, scaling, and rotations of objects performed outside of a *sub-level* of editing can all lead to these deformation issues, even if the model looks normal inside the modeling program.

The Digital Sky 2 environment can also support models with a very large file size. The International Space Station model which comes with Digital Sky 2 is a very detailed, large

model. The model can be displayed on the screen with no performance loss.

However, it is also important to know that when working with large file sizes, the development team will need to plan ahead on how to load these assets, since it will take time for a larger model to load than a conservatively smaller sized model. If there are also large textures associated with the model, this will also have an impact on file loading times.

C. Landscapes, Flat Surfaces, & Billboards

One of our major goals with the project was to create land-based scenery and events. This was a challenge due to the high field of view of our system. Flat objects were distorted to look curved. This resulted in a fisheye appearance to all of our content, and while it was acceptable for space scenes, it ruined landscapes and anything else we wanted to look 'flat'. Digital Sky 2 is set with a *field of view* associated with the connected display, and for us this editable value is set at around 170 degrees. However, it is a system-wide setting, and cannot be set on a per-scene basis. This led us to the conclusion that adjusting this setting will harm all our previously created shows, and was not an option.

Our billboard graphics, which are .JPG or .GIF images put directly onto the screen, also suffered from the fisheye problem. The further towards the top of the screen we put an image, the more distorted it was. We noted, that if we set a billboard low at the horizon, which was the bottom of the screen, it was flat, albeit cut off by the screen itself.

We discovered ideal ways to combat this distortion. The ideal display to use for billboards is *panoramic*. To shift a billboard upwards on the screen without distortion requires two values on two different lines to be changed. On the billboard setup line, the *Y-Offset* has to be an increasing positive value, and the display line has to keep the *elevation/declination* setting to a value close to zero (Figure 2).

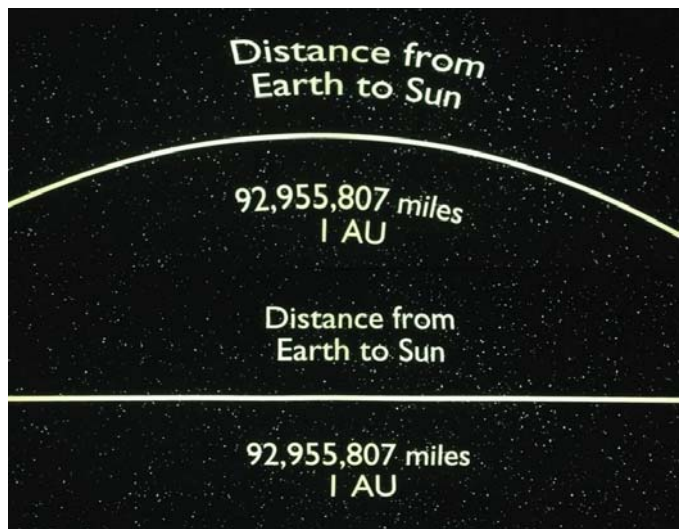


Figure 2: Top – No Y-Offset; Bottom – Y-Offset Used

Each billboard has different settings, since the size of the associated image is different. The *Y-Offset* has a maximum value it works at, so juggling between it and the *elevation/declination* value is required for an optimum image. Discovering that images placed along the horizon have very little bending to them helps to pave the way for landscape creation. If one creates a landscape by placing it at the base of the screen and pointing the camera 'skywards', the landscape will remain flat along the bottom of the screen.

Unfortunately this technique does not meet expectations when we want the camera to look at the landscape itself. When rotating the camera 'side to side', the fisheye effect is not very obvious. However, when rotating the camera 'up and down' the effect is dizzyingly prevalent. Due to the *field of view* setting, the screen is viewing an entire hemisphere.

Once one fully understands how the camera acts because of the field of view, one can come up with strategies to get around it in their models. What we discovered to work best was to create the scene we wanted, and then bend and distort it afterwards. By bending and extending the edges of the model to curve inwards, the camera will not be able to see the end of the landscape. It is also important to bend the model around where camera is to be located.

To build a good centerpiece terrain, it should be modeled from 'strips' of terrain, then overlapped them from the camera's point of view. This gives a good effect for distance. These strips should taper larger towards the edges to prevent them from appearing to shrink as they near the edges of the fisheye camera (Figure 3). This technique provides for excellent panoramic scenes.

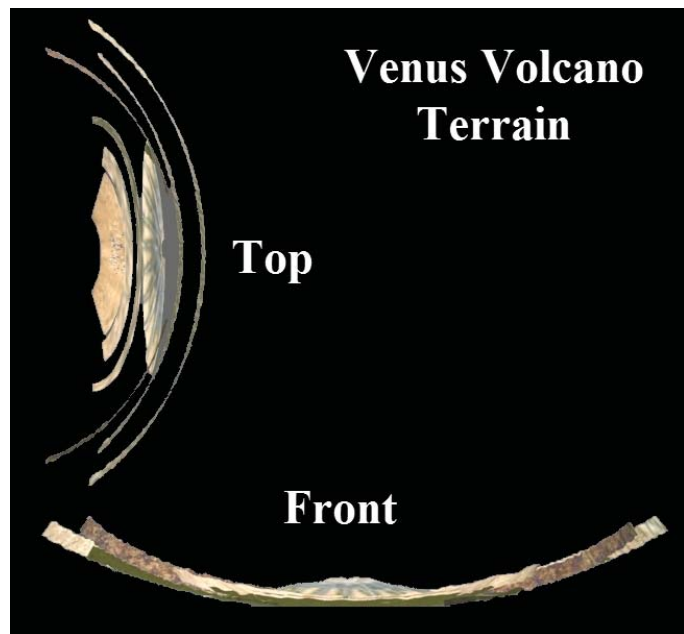


Figure 3: Terrain Bending for Camera Compensation

When building a scene it is important to know how the camera will act and what it will show to the audience. The design of the model will need to be based entirely how it is

seen by the camera. Once a scene has camera compensation bending applied to it, going back and making changes can be difficult, and even impossible for complex scenes.

D. Scene Animation

Digital Sky 2 allows for at least two types of animations: *bone-based* animations and the more standard *rotation and translation* animations. Bones animation involves the object being rigged with bones to animate it, and allows for more realistic animation [10]. The rotation and translation type animation allows for full objects to be rotated and translated, and may be useful for solid objects moving from one location to another location. Animations are built by setting *keyframes* for actions, which are fully implemented by Digital Sky 2.

For most complex animations, separating a static background model from an animated model is very challenging using Digital Sky 2 due to its own system complications. The primary reason to have two or more separate models in an animation is due to how the *shader* system works in Digital Sky 2. If a model is animated using bones with a basic bitmap applied to it, the bones will animate as expected. If that same model has a *shader* applied to it, the model will 'break apart' at the polygon level as the bones animate. This means that instead of having the polygons bend and shift to accommodate the movement of the bones, they are instead static and unable to shift, causing the model to literally rip apart at the seams as it animates. What this results in is static models that can have nice looking *shaders*, but any animated model must use only standard bitmaps. Being able to mix an animated, bitmap-only model over a nicely *shader-enabled* static model would be useful, however, actually getting this to work proved to be more of a challenge than we could afford given our time limit for the project. Mixing static and animated models together is possible, but it takes a considerable amount of work, and any changes to the animation or static scene has adverse effects on the Digital Sky 2 settings and positions of the models.

For optimum control over a fully animated scene, it is best to keep the animating objects in the same model as their background. All of the scene objects must also be connected together as a single object, or there may be unexpected results. With a single large object, the only way to animate it is by using the bones system. Surmounting object interaction and positioning is difficult, and using this technique for precision will save countless hours for different animations.

It is important to keep in mind that Digital Sky 2 mirrors everything when it loads it into its environment - just like models. So an animation of something going left will go right inside the Digital Sky 2 environment. This can be circumvented by mirroring the scene prior to exporting.

E. Applying Bitmaps and Shaders

Creating a quality model is a two-part event. First, a detailed model must be developed, and second, good bitmaps and shaders must be applied to it. Digital Sky 2 allows these features with shader support available for non-bones animated objects. Getting a basic bitmap on an object requires nothing more than exporting the bitmapped model with the PandoSoft

exporter. Applying a shader to the model requires significantly more work.

To apply a shader to a model, first one must export the model as normal using the exporter software, like Pandasoft. The shader file itself will need to be created separately. Shader files have a .fx extension, and pre-existing .fx files can be found within Digital Sky 2 to use as a starting point [11]. Digital Sky 2 supports many types of *shaders* that are not normally available within the default .fx files, such as self-illumination. These shader additions can be built into custom .fx files if necessary for a particular model.

With either a custom shader or a pre-built shader from another Digital Sky 2 file, one must then connect the shader with the model. This requires editing the .x model file. To best understand how this works, one can investigate the *Shuttle.x* and *StandardFX_ds2.fx* files which come with Digital Sky 2. *StandardFX_ds2.fx* by itself can handle *normal* mapping, *specular* mapping, and *diffuse* mapping techniques.

To apply the .fx file to the model, open the .x file in an editor and search for *MeshTextureCoords*, locate the texture method above that block. If this is a fresh export using the Pandasoft exporter, there will only be the bitmap pointing to an image file. In order to get the *StandardFX_ds2.fx* shader applied within model, replace the entire bitmap block with the contents of the block in the same location as shown in the *Shuttle.x* file. To use custom bitmaps, change the file names pointed to in the *Shuttle.x* block.

If the model is using multiple objects inside of a single file, each object will need to have the bitmap block replaced with the associated shader block. This can become time consuming, but it is possible and it will allow for different objects within a scene to use different shaders. Good usages for multiple objects in a single file would be for such things as a space station and a space ship docked together. Instead of trying to wrestle with the Digital Sky 2 environment for precision, simply merge the two objects into a single file and position them as a single model.

Another important note concerning mapping and Digital Sky 2 is that any model will be mirrored once it is inside the environment. This means that readable text inside the modeling program will be backwards inside of the Digital Sky 2 environment. This issue can be circumvented by mirroring the model prior to exporting.



Figure 3: Self-Illumination Shader Bitmap

We discovered that many 3D-engine specific shader functions work with the Digital Sky 2 engine, such as self-illumination. Self-illumination bitmaps need to have the diffuse bitmap edited to where the light is shining and black where there is no light (Figure 3).

The shader system used by Digital Sky 2 does not provide any documentation. However, we have observed that shader equipped models can be built to the level of current generation graphical expectations. Many of the same concepts used by high profile video game models can be employed within the Digital Sky 2 environment as well [12].

F. Large Project Development

Larger projects have their own development dynamics. Since Digital Sky 2 does not feature any way to speed up or slow down the speed of a show, a developer must sit through the entire segment before they can see the effect of their changes. This issue takes center stage as sound components are added to a presentation.

One way to combat this issue involves creating event-specific buttons on the control program. If a single button controls an exceptionally long segment, one can break that segment up into two or more buttons. Good split locations are most often camera change events. If one copies the last camera change made into the top of the new event button with a time value of zero, the camera will start at the end of the previous segment.

By using this technique, it has cut our own development time considerably. When we add sounds or make small changes to the movement of the camera or an object, we no longer need to wait for 20 minutes before we see the impact of that change. In other words, once editing is complete, one can cut and paste the fragment back into the original button event, and remove the camera position line.

When we develop large segments, we develop them as small partitions from the outset. Once all of the partitions are complete without error, we transfer them to a final button as a single event.

An important element to keep in mind when using this design method is locality. For example if segment 12 is being edited and requires elements from segment 2 to exist, there may be complications due to these missing elements. Ensure that all of the resources a segment requires have been loaded and are in their proper positions for that segment's time frame.

G. Loading Time and Memory Management

As a show is running, one hopes that the audience will become immersed in the newly created world. Suddenly, the picture stutters, and the audience is jolted back into reality. Or even worse, one of the computers crashes, and suddenly the audience is watching a Windows desktop. These issues arise from asset loading and poor memory management. For large, ambitious projects featuring considerable custom content, loading times and memory management become a problematic hurdle to overcome.

Currently in Digital Sky 2, when an asset loads, it is loaded in the background. This loading can be almost seamless for small billboard images or sound bites. However, when the system needs to prepare for a scene shift, and a large number of assets are loading, this a noticeable on-screen hiccup. This problem can also occur for a small model if it happens to use *shaders*, since it may have four or more large bitmaps to load before it can be displayed. Hiccups can also occur when loading a large video file for playback. In fact, any large file has the potential to impact the quality of the on-screen frame rate.

One of the best techniques to combat this behavior is to load assets when the camera is stationary. The frame rate will still be impacted, but because the camera is not actually moving the loss of frames will not be noticed. Using a fade to black type transition also makes use of this technique. A fade to black also looks more professional than simply switching from one scene to another instantly.

The least effective method to utilize is to gradually load assets. For instance, in the feature's code, one can add *keyframes* between camera positions which do nothing but load assets. In this case, loading assets one at a time over a period of time may not result in a noticeable frame rate impact. For example, if a camera moves over a 20-second period, adding additional *keyframes* at 3 second intervals will not affect the movement of the camera or how long it takes to move. However, one should also make sure that the final load takes place before the initial 20 second camera movement has completed.

Digital Sky 2 has no automatic memory management features. This means that for every asset loaded, it must be unloaded or it will continue to take up memory until the system is reset or crashes. When a crash takes place, one of the computers usually will shut down Digital Sky 2 and return to the desktop. This problem may arise when playing many concurrent shows, since one of those shows may have a

memory leak. A general good rule of thumb is to hit the reset button between shows to prevent any unexpected behavior.

To combat this problem and keep a show running smoothly, assets should be unloaded when they are no longer used. If an asset is used more than once for different scenes, hiding it may be more useful than unloading and reloading it.

At the completion of a show make sure all assets are completely unloaded. Breaking a large feature into segments will make memory management easier to handle, since loading and unloading can take place on a segment-by-segment basis.

H. Building an Interactive Presentation

The interactive feature is the reason why we began working on this project. Surprisingly, it is simple to accomplish with Digital Sky 2. Not only are the event buttons useful for creating a single large feature, but they can also be used to control an interactive show.

The buttons can mirror what the storyboard overview depicts. Each button can control that particular segment from the storyboard. The show's presenter will then be able to easily follow the flow of the show from one button to the next.

Helper buttons may also be implemented. We use helper buttons for naming the spacecraft of our show to what the students chose. This involves us needing to edit that button during the show, which contains the code to give a name to an object already loaded on the screen, and then activate that button. The students enjoy seeing that they could really interact with and change what was visually on screen. Being able to give key objects custom names helps to keep the show personal and individualized.

IV. CONCLUSIONS

The ending product for our project was very close to our vision for graphical fidelity. Using the techniques we learned, we were able to use Digital Sky 2 to seemingly bring people not only to the far reaches of space, but also to the surface of Venus and Callisto. We were able to create branching events and tie it all together to create a new kind planetarium presentation. Our success was the approval of teachers as well as the interest of the students. Our premier show aired all day long to nearly 300 people, and the show has been added to the regular schedule for schools visits. Using the tools and techniques described in this paper, we have successfully brought student interest into an area that previously had very little interaction.

Measuring the success of a public show, where no grades or surveys are involved, can be a difficult task. However, most educators can recognize the difference between students that are truly interested versus those that are "just attending". Following the premier and consequent presentations of our pilot interactive feature, we observed more inquisitive students than we had previously. There was also more of a buzz about their experiences during lunches, and more activity between "project zones". When the general target audience is middle school children, data gathering can be anything but simple and often inconclusive. We are therefore left to our own

observations of the students' reactions, such as smiles and eagerness.

If nothing else, we hope that our own success and direction provided in this paper will help other educators create an exciting educational environment. Even if one is using something other than Digital Sky 2, this paper still holds valuable resources for an educator to outline and create a larger production. It is not easy, but capturing the minds and sparking the curiosity of students and adults alike is a reward only measurable by our future society.

V. FUTURE WORK

With the success of our project, the Millard Oakley STEM Center on the Tennessee Tech University campus is actively looking to continue to create new interactive content. We know there is more tweaking and tricks that can be utilized by the Digital Sky 2 system. For instance, one of the tricks we have been thinking about has been a way to employ a Java program to cause 'on the fly' edits to a script in the Digital Sky 2 system. This can allow for students to interact with the system via a computer set up in the room for them, and it would remove the necessity for an operator to script in whatever custom event a student would build. One of the tweaking techniques we are currently looking into is the reason behind a model breaking during bones-based animation if it has a shader applied to it as opposed to a regular bitmap. We are investigating the impact that other files within the system may be having on this issue. We are also still investigating the unknown object physics that can cause two models to 'collide' with each other and move each other out of position. This appears to happen at random, and as of the writing of this paper, we have been unable to determine how to compensate for this behavior by the system. We are hoping to investigate this problem further and discover a workable solution.

VI. ACKNOWLEDGEMENTS

This project could also have not been possible without Marc Robinson, who was the co-creator of the presentation

from conception to giving the show to students. This project would also not have been possible without Dr. Sally Pardue, who has incredible dedication to STEM activities and is always looking for ways to improve student learning and experiences.

REFERENCES

- [1] "Expedition field trips for schools." Tennessee Technological University. <http://www.tntech.edu/stem/expedition-field-trips/> Accessed: 3 March 2013
- [2] Sky-Skan. <http://www.skyskan.com/products/systems> Accessed: 3 March 2013
- [3] Morehead Planetarium and Science Center, The University of North Carolina at Chapel Hill. <http://www.moreheadplanetarium.org> Accessed: 7 March 2013
- [4] The Lawrence Hall of Science, University of California, Berkeley. <http://www.lawrencehallofscience.org/visit/activities/planetarium/about> Accessed: 7 March 2013
- [5] "Storyboarding." Stanford University. <http://acomp.stanford.edu/tutorials/storyboarding> Accessed: 18 March 2013
- [6] "X File Reference (Legacy)." Microsoft MSDN. (2013). <http://msdn.microsoft.com/en-us/library/windows/desktop/bb173011%28v=vs.85%29.aspx> Accessed: 9 March 2013
- [7] Blender. <http://www.blender.org/> AccessedL 9 March 2013
- [8] "3D Studio Max." Autodesk. <http://usa.autodesk.com/3ds-max/> Accessed: 9 March 2013
- [9] "Panda DirectX Exporter." Pandasoft. http://www.andytather.co.uk/panda/directxmax_downloads.aspx Accessed: 10 March 2013
- [10] "OpenGL: Tutorials: Basic Bones System." Gpwiki.org. http://content.gpwiki.org/index.php/OpenGL:Tutorials:Basic_Bones_Sy stem Accessed: 22 March 2013
- [11] "Effect (.fx) Files." XBDEV.NET. <http://www.xbdev.net/shaderx/fx/index.php> Accessed: 12 March 2013
- [12] "Modeling Techniques: Movies vs. Games." ACM Siggraph. <http://www.siggraph.org/publications/newsletter/volume-41-number-2/modeling-techniques-movies-vs-games> Accessed: 27 March 2013