CAMERA PLACEMENT UTILITY FOR DIALOGUE SEQUENCES

A Thesis

by

RYAN DAVIS SHARPE

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Chair of Committee,	Ann McNamara
Committee Members,	Carol LaFayette
	Daniel Humphrey
Head of Department,	Tim McLaughlin

May 2019

Major Subject: Visualization

Copyright 2019 Ryan Davis Sharpe

ABSTRACT

Cinematography is critical in communicating visual ideas. In narrative driven games, dialogue sequences have a common set of camera shots that are repeated over the sequence such as the interior, exterior, and apex shots. I propose a tool that procedurally places these shots allowing anyone with limited cinematographic knowledge to create dialogue sequences easily. Procedural placement also reduces the need for manual placement and speeds workflow in constructing shots. In addition, the tool also allows the user to define their own shots to be procedurally placed allowing further customization in creating a dialogue sequence.

CONTRIBUTORS AND FUNDING SOURCES

This work was supervised by a dissertation committee consisting of Professor Ann McNamara and Professor Carol LaFayette of the Department of Visualization and Professor Daniel Humphrey of the Department of Liberal Arts.

All work for the thesis was completed independently by the student. No other external contributions were made.

NOMENCLATURE

LOD	Line of Dialogue
LOA	Line of Action
PSO	Playable Script Object
UI	User Interface

TABLE OF CONTENTS

	Page
ABSTRACT	ii
CONTRIBUTORS AND FUNDING SOURCES	iii
NOMENCLATURE	iv
TABLE OF CONTENTS	v
LIST OF FIGURES	vii
1. INTRODUCTION	1
1.1 A Tool for a Non-Cinematographically Familiar Audience	1
2. LITERATURE REVIEW	2
2.1 Challenges & Solutions in Automated Cinematography Tools	2
2.2 Frameworks Utilizing Narrative Information	3
2.3 Relevant Techniques to Determine Camera Placement	5
3. METHODOLOGY & DESIGN	8
3.1 Proposed System	8
3.2 Functionality Overview	8
3.3 Shot Calculation Classes	9
3.3.1 CameraShot	10
3.3.2 FrameShare	12
3.3.3 OverShoulder	15
3.4 Maintaining the Line of Action	17
3.5 UI Design Overview	18
3.5.1 Adding Cuts and the Opposite Option	20
3.5.2 Custom Shot Creator and Editor	21
3.5.3 Dialogue Objects and Assigning Audio & Animation	22
3.5.4 Exporting Shots, Dialogue Text, Audio, and Animation	23
3.6 Saving Shot Patterns	23

4. CONCLUSION & FUTURE WORK 25

4.1 Conclusion 25
4.2 Future Work 25
4.2.1 Extending CineCam 25
4.2.2 Crowdsourcing Cinematographic Styles 26
4.2.3 Allowing for Different Character Heights 27

REFERENCES 28

LIST OF FIGURES

Page

Figure 1: Reprinted from Lai example output of a shot from their system 3

Figure 2: Reprinted from Tomlinson an example output from their system 4

Figure 3: Several examples of CameraShot Specifications, in order Default, Default-Close, Default-Mid, Parallel, Low Angle, High Angle. 11

Figure 4: Visual representation of the CameraShot shot specification algorithm. 12

Figure 5: FrameShare shot. 14

Figure 6: Visual representation of the FrameShare shot specification algorithm........................ 14

Figure 7: OverShoulder shot. 16

Figure 8: Visual representation of the OverShoulder shot specification algorithm. 16

Figure 9: Default composition determined by left or right side.. 17

Figure 10: Visual representation of CineCam maintating the line of action with the left and right side marker.. 18

Figure 11: The Complete User Interface of CineCam. 19

Figure 12: The Cuts Panel Interface. 20

Figure 13: Custom Shot Creator Panel. 21

Figure 14: Edit Shot Panel. 22

Figure 15: Adding Audio & Animation Panel. 22

Figure 16: Previz and Export Buttons.. 23

1. INTRODUCTION

Cinematography is a key component of modern computer games as the way a scene is presented to the player plays a critical role in their perception of the story. There has been an interest in developing systems and methodologies that automate camera placement so as to remove or reduce the need for a cinematographer. This thesis specifically focuses on cinematography in dialogue contexts. Games like “Mass Effect” and “Life is Strange” have many dialogue sequences in them to advance the narrative. Dialogue scenes commonly have variations of the interior, exterior, and apex shots. Procedurally placing these shots instead of manually placing them with a tool would be beneficial in speeding workflow. Previous research efforts explored systems that procedurally compose shots from narrative information. However, I do not believe these types of systems are effective because capturing the intent of the cinematographer is difficult to automate with an engine or AI.

1.1 A Tool for a Non-Cinematographically Familiar Audience

Game development is a very multidisciplinary and evolving field. Often a developer can be involved in roles that go beyond their expertise. Creating tools that take into account the limitations of knowledge of the user is beneficial in speeding workflow. I have designed a camera placement tool for a non-cinematographically familiar audience. With the system’s UI, and built in cinematographic knowledge, anyone using the tool can create a dialogue sequence easily.

2. LITERATURE REVIEW

2.1 Challenges & Solutions in Automated Cinematography Tools

Throughout the field of automated cinematography, there have been several avenues where automated camera placement has been explored. This includes real-time camera placement, as well creating a formalized language for camera placement.

Previous research has been done in automating camera placement in real-time within dynamic environments. In video games, the player often is in control of the camera. Burelli's research noted that player control often conflicts in an automated camera system, as an automated camera takes control away from the player, making the game less playable at the expense of making the game look more cinematographic [8]. This is one of the many challenges in real-time automated systems.

Bares et al. focused their research on the challenges of real-time camera control in dynamic environments as well. They proposed a system called UCAM that employs "cinematographic user models to render customized visualizations of dynamic 3D environments" [9]. Essentially UCAM's cinematographic adaptive models change as the environment changes, enabling the camera to visualize the environment in a clear way independent of where it is in the environment.

Though solving the automated camera placement in real-time contexts has been a focus, formalizing a language for camera placement has also been explored. In the research done by Chistianson et al, they proposed encoding cinematographic principles into a formalized language

in “Declarative Camera Control for Automatic Cinematography” [10]. The encodings written in the language is fed into a system which has a Hueristic evaluator that places the camera in the scene. This formalized language is useful for engines or systems implementing camera positions.

2.2 Frameworks Utilizing Narrative Information

Many previously proposed procedural camera placement systems often have narrative engines that analyze a story or a script to determine shot placement. Pei-Chun Lai et al in their paper proposed a pattern-based tool for creating virtual cinematography [6]. In their system, the story context is inputted as XML parameters that include Scene, Emotion, Communicative Goals, Actions, and Character elements. The system matches these story elements with camera patterns and outputs a shot sequence. After viewing an example of a shot sequence outputted by their system, I noticed the scene did not cinematographically flow well. This speaks to the difficulties in automating artistic decision making.

Figure 1: Reprinted from Lai example output of a shot from their system [6]

In Tomlinson et. al's system "Expressive Autonomus Cinematography for Interactive Virtual Environments", narrative information is encoded into the scene. The characters have encoded emotion information like happy, sad, angry, and surprised. The scene also contains motivation information and it is characterized as "DesireForTwoShot", and "DesireForCloseUp". The system extracts the emotion and motivation information as well as the height and position of the characters into a behavior tree. The behavior tree consists of predefined shots and uses this information to make decisions on shot placement [11].

Figure 2: Reprinted from Tomlinson an example output from their system [11].

In Tomlinson et. al's assessment of their system, they noted that when participants used their system, sometimes the character would travel offscreen and the participant would be confused. They determined when the participant was confused when participant would audibly

tell the person conducting the assessment that they could no longer see the character. Their system highlights the general challenge in using narrative information for autonomous camera behavior.

Another tool not directly related to camera placement but similarly uses story or script to inform decision making is Disney Research's Cardinal. Cardinal provides a means of viewing a script through a variety of perspectives [7]. Cardinal uses natural language processing to visualize traditional movie scripts. "The system takes the text and stores it in an internal meta-annotated representation format. Text of actions is further passed to a natural language processing server and transformed into other visualizations" [7]. Cardinal can create an interaction view, which visualizes the actors as set lines and groups the lines in terms of interactions the actors have. Cardinal also offers a 3D view that uses the ADAPT framework which uses behavior trees to determine character animations [7].

The Cardinal System shows that using narrative information to create a visual output can be useful if utilized in a correct context. In this case, the goal of Cardinal is to visualize the flow of actions in the script and not necessarily create a production ready representation. In this sense, Cardinal is successful in creating a useful result from narrative information as their end result is a preliminary.

2.3 Relevant Techniques to Determine Camera Placement

Previous efforts in camera placement systems have used various techniques to determine camera placement. Three noteworthy techniques are Director's Volumes, idioms, and constraints.

The Director's Volume technique as proposed by Lino et al computes spatial partitions in the space of viewpoints around a key subject. "Each partition is qualified with a semantic tag representing its shot distance and relative angle to subjects" [03]. Categorization of regions of space around the subject is useful for a system to determine what is a close shot versus a mid-shot.

Idioms are a convenient and straightforward way to encode cinematographic knowledge as mentioned by Cozic [01]. They are essentially hard-coded shot specifications. They are very useful for capturing cinematographic techniques accurately, however they become difficult to adapt to dynamically changing virtual scenes. Studies by Galvane et al stated that idiom-based techniques would typically fail in this area due to the inability of the technique to handle complex situations and the necessity arises to design idioms for many situations [02].

The solution to drawbacks of idioms are constraint-based solutions. Given a set of constrains about the objects to appear in the frame, the system tries to find the camera parameters that best satisfy the constraints as seen in Friedman's proposed system [04]. Constraints are essentially a set of rules that allow the camera to react in different ways depending on the situation or condition. For example, if the character entered a new area, and the scene needed be viewed and played from a bird-eye view, the camera could be positioned at the top of the scene and be constrained from rotating while following where the player travels. Previous systems have applied constraints as rules of editing [05]. Rules such as continuity of style, line of action, screen continuity, and motion continuity.

Considering these limitations and the techniques used in other systems, I propose an automated camera system specifically focused on dialogue sequences. This system has the capability to procedurally frame compositions accurately within the dialogue context.

3. METHODOLOGY & DESIGN

3.1 Proposed System

The proposed system CineCam, is a camera utility exclusively designed for dialogue sequences. Sequences involving dialogue have common camera cuts such as variations of apex, interior, and exterior. Unlike a real-time camera context, the dialogue scene context lends itself well to being automated as dialogue shots have the potential be procedurally placed accurately. Unlike other systems, CineCam does not use a script or narrative engine to inform camera placement but instead uses a UI to allow the user to select shots. CineCam is designed to be a utility to reduce manual camera placement.

CineCam stores camera definitions as idioms. These idioms are associated with different cinematographic techniques. CineCam's user interface (UI) allows the user to select and place a shot. They can also edit any shot by changing the idiomatic variables associated with the selected shot. The tool also allows the user to create their own custom idioms, giving the user flexibility in composing shots.

As previously mentioned, the user in most cases might not be cinematographically familiar. When the user creates a custom shot, it is based on a pre-existing shot definition. This allows them to create shots that are approximately accurate to commonly found shots found in dialogue sequences.

3.2 Functionality Overview

A script with the actors and their corresponding lines of dialogue is read by the CineCam system and assigns a default camera shot for each line of dialogue. The UI displays the

resulting actor name, dialogue, and camera shot. Additional shots can be added with the "Add Cut" button and removed with the "X" button. This allows for multiple cuts to happen during a single line of dialogue. A dropdown menu allows the user to select the type of shot from a list. The user can choose from various cinematographic techniques such as "High Angle", "Low Angle", "Overshoulder", "Parallel", and "FrameShare". The user can additionally assign audio and animation that the character is saying for that LOD.

The resulting shots, audio and animation data can be exported into a PSO (Playable Script Object). The PSO contains all the relevant sequence information such as shot idioms, dialogue text, dialogue audio, animations, and the location of the characters in the sequence. When the PSO is accessed and called in the script through a public function, the sequence is played.

The custom shot creator which is the second aspect of the tool, allows the user to create and define their own custom shot. The custom shot creator has a preview windows that displays what the shot will look like. When a custom shot is created, it can be saved as an idiom and can be added through the dropdown list and applied on any character. The custom shot creator also allows the user to further edit current placed shots. For example, the user could take the Default shot, tweak the values, rename it and save it as a custom shot.

3.3 Shot Calculation Classes

There are three types of shot classes: CameraShot, FrameShare, and OverShoulder. Each class has its own implementation and calculation for composing the camera around the actor or

actors. Distance, height, orbit, and Xbias are variables that are shared in the calculations of each of these shots. Distance is the length away from the actor, height is the length above the actor, orbit is the degree of rotation around the actor, and the Xbias is the distance the camera shifts on its local x axis, framing the actor on either the right or left side of the screen.

3.3.1 CameraShot

The CameraShot class is used to create interior shots and variations of an interior shot. These include mid-shots, closeups, extreme closeups, as well as high angle and low angle shots. It is used for when the focus is on a singular character. The other character is irrelevant in calculating the composition of the shot. Figure 1 shows variations of the CameraShot specification.

The following Unity C# code and vector math is the algorithm the CameraShot class uses to calculate a CameraShot shot.

```
//Set Camera Distance Away From Target
Vector3 targPos = targetObj.transform.position;
CamPos = targPos;
Vector3 forwardN = (targetObj.transform.forward).normalized;
CamPos = CamPos + (forwardN *  distanceFromTarget);

//Adjust Camera Height to Go up or Down
CamPos = new Vector3(CamPos.x, CamPos.y + height, CamPos.z);

//Calculate Camera Rotation
GameObject cam = new GameObject();
cam.transform.position = CamPos;
cam.transform.RotateAround (targPos, Vector3.up, orbitAngle);
Vector3 option1 = cam.transform.position;
cam.transform.position = CamPos;
cam.transform.RotateAround (targPos, Vector3.up, -orbitAngle);
Vector3 option2 = cam.transform.position;
CamPos = GetClosest(sidemarker, option1, option2);
```

```
//Look Directly at Target
CamRot = Quaternion.LookRotation(targPos - CamPos);
//Apply the Bias Shift
cam.transform.position = CamPos;
cam.transform.position += Vector3.forward * biasX;
option1 = cam.transform.position;
cam.transform.position = CamPos;
cam.transform.position += Vector3.forward * -biasX;
option2 = cam.transform.position;
CamPos = GetFarthest (sidemarker, option1, option2);

UnityEngine.Object.DestroyImmediate (cam);
```

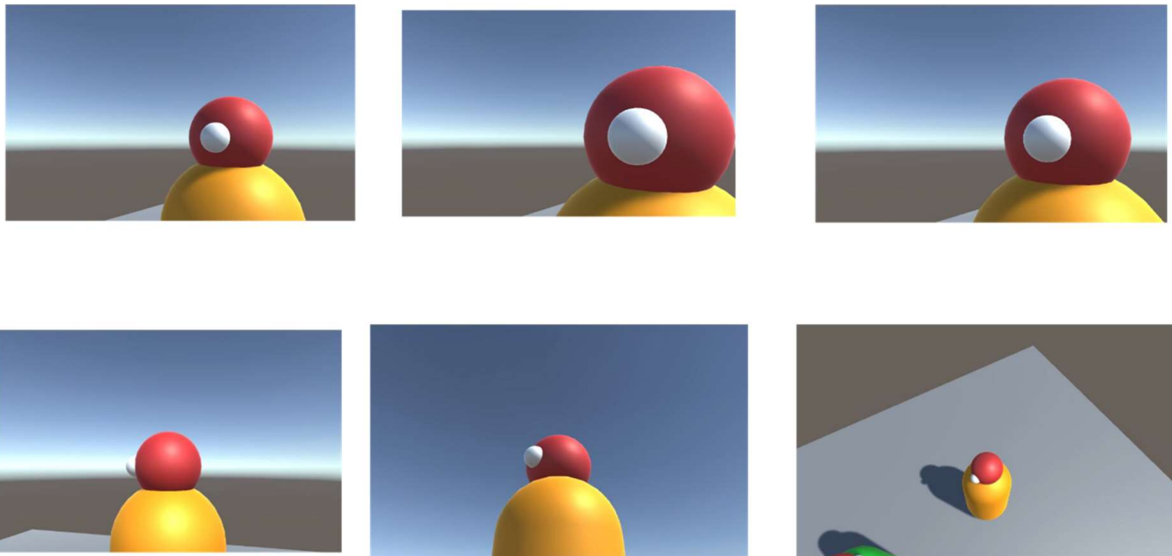

Figure 3: Several examples of CameraShot shots. An implementation of the interior shot. The shots are in order defined as Default, Default-Close, Default-Mid, Parallel, LowAngle, HighAngle.

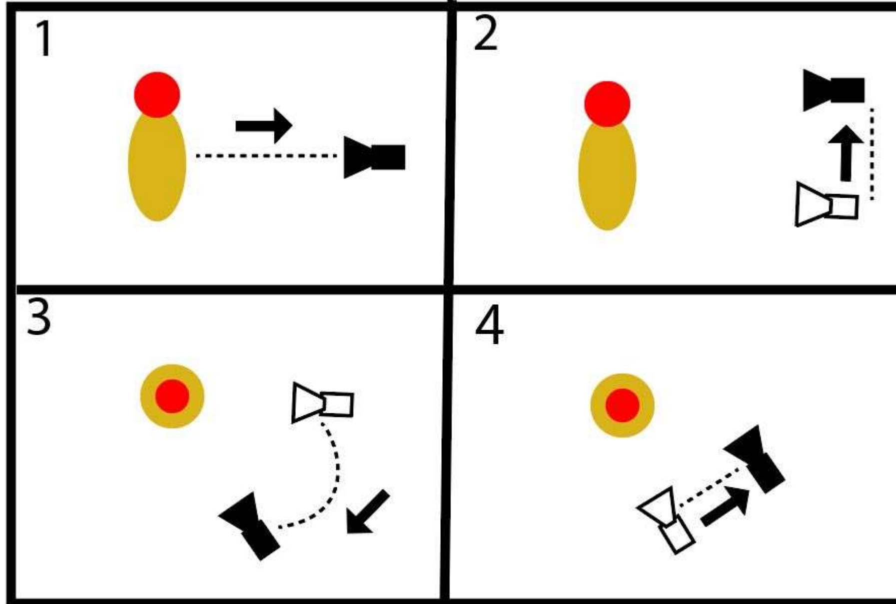

Figure 4: Visual representation of the CameraShot shot specification algorithm.

3.3.2 FrameShare

The FrameShare class is used to create apex shots and variations of the apex shot. An apex shot is a shot that includes both actors as seen in Figure 3. FrameShare shots are calculated by finding the midpoint between two actors. The camera is oriented around that midpoint based on the distance, height, Xbias, and orbit values. FrameShare shots are effective in conveying affinity between two actors. They are commonly used in romantic dialogue scenes as well as scenes where two actors are communicating something that is of mutual value.

The following Unity C# code and vector math is the algorithm used by the FrameShare class to calculate a FrameShare shot.

```
//Calculate Camera Position distance
cam.transform.position = targetObj1.transform.position;
float actorDistance = Vector3.Distance(targetObj1.transform.position,
targetObj2.transform.position);
```

```
//Get Direction of Actor1 towards Actor2
Vector3 actorADirN =  (targetObj2.transform.position -
targetObj1.transform.position).normalized;

//Midpoint in actor direction
Vector3 MidPoint = targetObj1.transform.position + (actorADirN * (actorDistance / 2));

//Calculate Perpendicular vector
//Rotate ActorADirN 90 degrees on y axis
Vector3 PDir1 = Quaternion.AngleAxis(90, Vector3.up) * actorADirN;
Vector3 PDir2 = Quaternion.AngleAxis(-90, Vector3.up) * actorADirN;

Vector3 option1 = MidPoint + (PDir1 * distanceFromTarget);
Vector3 option2 = MidPoint + (PDir2 * distanceFromTarget);

CamPos = GetClosest (sidemarker, option1, option2);

//Adjust height to go up or down
CamPos = new Vector3(CamPos.x, CamPos.y + height, CamPos.z);
cam.transform.position = CamPos;

//Orbit around Midpoint
cam.transform.RotateAround (MidPoint, Vector3.up, -orbitAngle);

//Look At MidPoint
Vector3 groupDirN = (MidPoint - cam.transform.position).normalized;
Quaternion rotation = Quaternion.LookRotation(groupDirN);
cam.transform.rotation = rotation;

CamPos = cam.transform.position;
CamRot = cam.transform.rotation;
UnityEngine.Object.DestroyImmediate (cam);
```

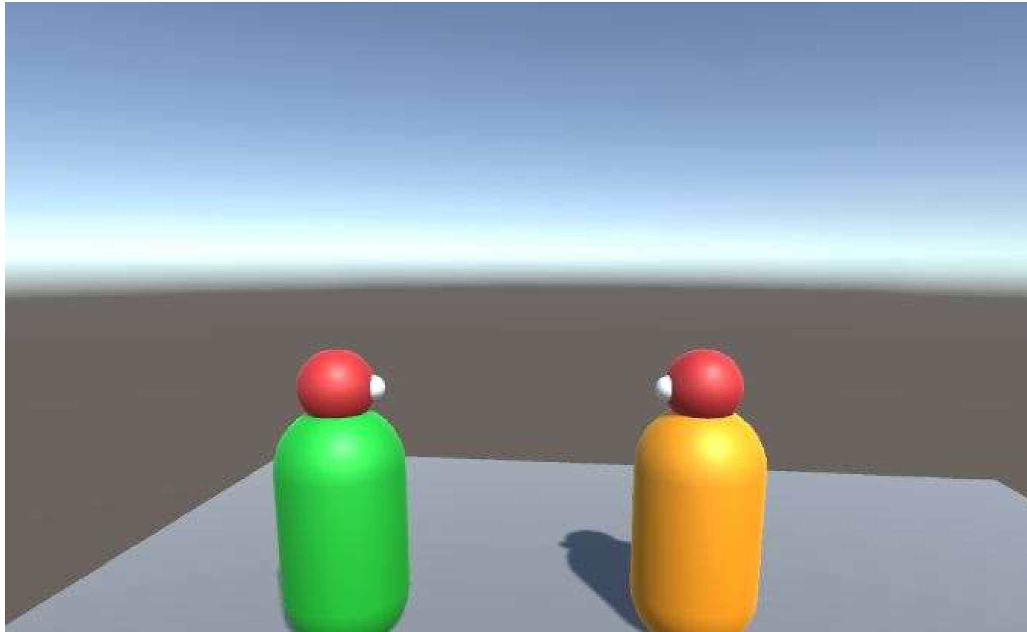

Figure 5: FrameShare Shot, an implementation of the apex shot. Both characters share screen space equally.

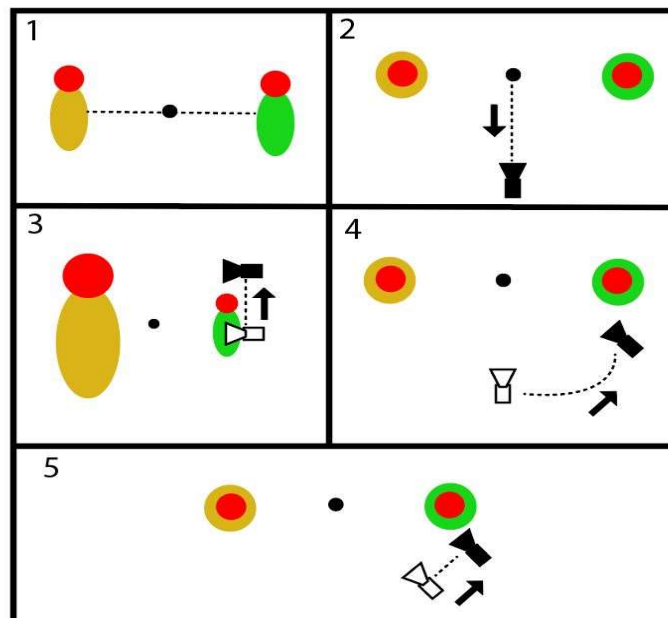

Figure 6: Visual representation of the FrameShare shot specification algorithm

3.3.3 OverShoulder

The OverShoulder class is used for shots when the focus is on the speaker but the recipient also needs to be in the frame. The shot is calculated by orienting the camera on the recipient, then it is rotated towards the speaker so the speaker can be seen over the recipient's shoulder. Like CameraShot and Frameshare, the properties of this shot can be changed with the distance, orbit, Xbias, and height variables. Figure 5 shows an example of an OverShoulder shot.

The following Unity C# code and vector math is the algorithm carried out to calculate an OverShoulder shot.

```
Vector3 targPos = targetObj.transform.position;
CamPos = targPos;
Vector3 forwardN = (targetObj.transform.forward).normalized;
CamPos = CamPos + (forwardN *  distanceFromTarget);

//Adjust height either up or down
CamPos = new Vector3(CamPos.x, CamPos.y + height, CamPos.z);

//Calculate Camera rotation
GameObject cam = new GameObject();
cam.transform.position = CamPos;
cam.transform.RotateAround (targPos, Vector3.up, orbitAngle);
Vector3 option1 = cam.transform.position;
cam.transform.position = CamPos;
cam.transform.RotateAround (targPos, Vector3.up, -orbitAngle);
Vector3 option2 = cam.transform.position;

CamPos = GetClosest(sidemarker, option1, option2);

//Look Directly at Target
CamRot = Quaternion.LookRotation(targetObj2.transform.position - CamPos);

UnityEngine.Object.DestroyImmediate (cam);
```

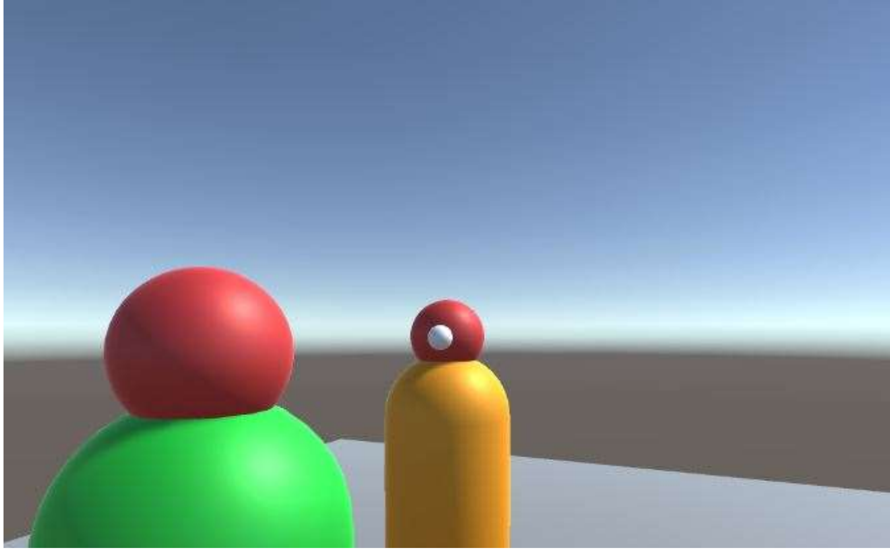

Figure 7: OverShoulder Shot. An implementation of the exterior shot. Another character included in the composition.

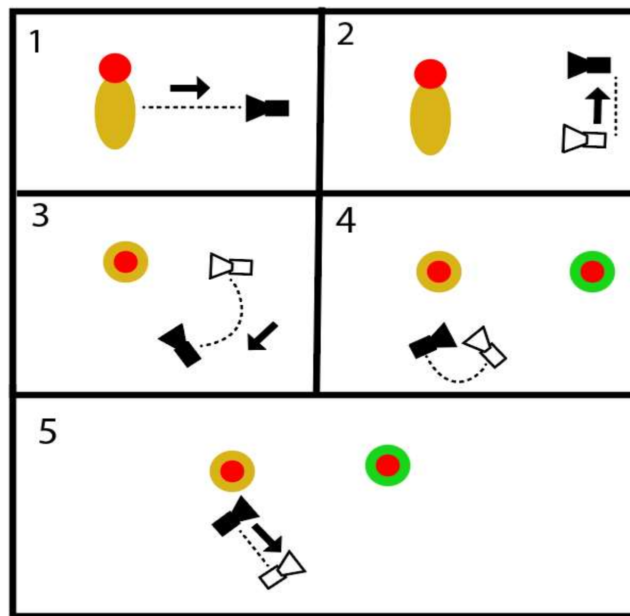

Figure 8: Visual representation for the OverShoulder shot specification algorithm

3.4 Maintaining the Line of Action

All shots maintain the line of action (LOA) also known in cinematography as the 180-degree rule. Maintaining the LOA means that the shots always stay on the left or right side of the actors and do not cross the axis between the two actors. Crossing the LOA often can be disorienting and it can distract the audience from the intent of the scene.

When CineCam is run, it checks which side has been selected (right or left). Then a SideMarker is placed which is a vector variable that contains coordinates for a position in the 3D space on the left or right side of the actors. When the orbit angle of a shot is calculated, it orbits in the direction closest to the SideMarker. This allows for the shots to maintain the LOA and stay visually consistent. The dropdown menu allows the user to select either the left or right side to change which side of the actors the shots are being calculated on.

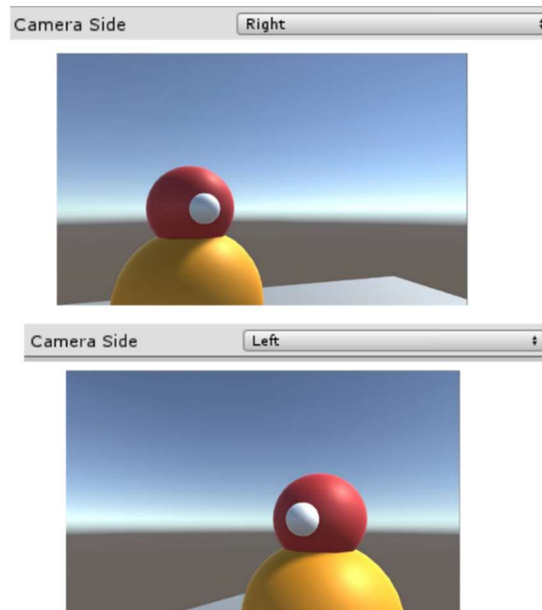

Figure 9: Default composition determined by left or right side toggle

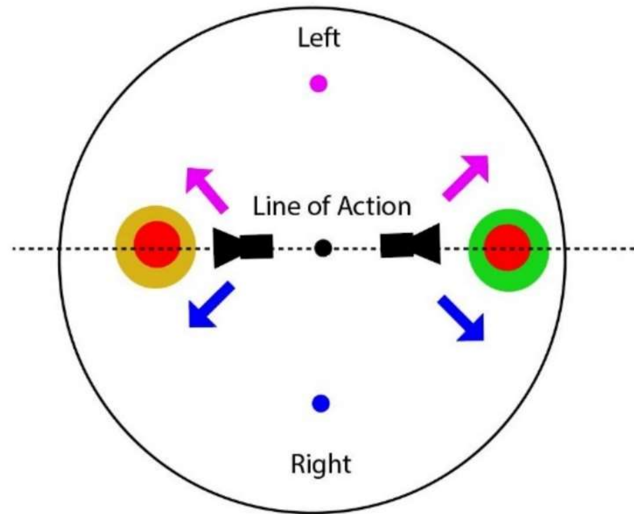

Figure 10: Visual representation of CineCam maintaining the line of action with the Left and Right sidemarkers

3.5 UI Design Overview

Implementation of the CineCam system is done through the CineCam UI. The UI is separated by four main sections. The Actor, Dialogue, Shots, and Audio / Animation as seen in figure 9. In the middle of the UI are the export and previsualization buttons. On the bottom of the UI is the shot customizer where the user can edit current shots and create their own. The window on the bottom right previews what the camera composition looks like when a shot is being edited or created.

The goal of the design of the CineCam UI is to clearly display actor and dialogue information as well as shot information so the user can modify and arrange and edit shots. The dialogue and shot information are listed the order of the dialogue script.

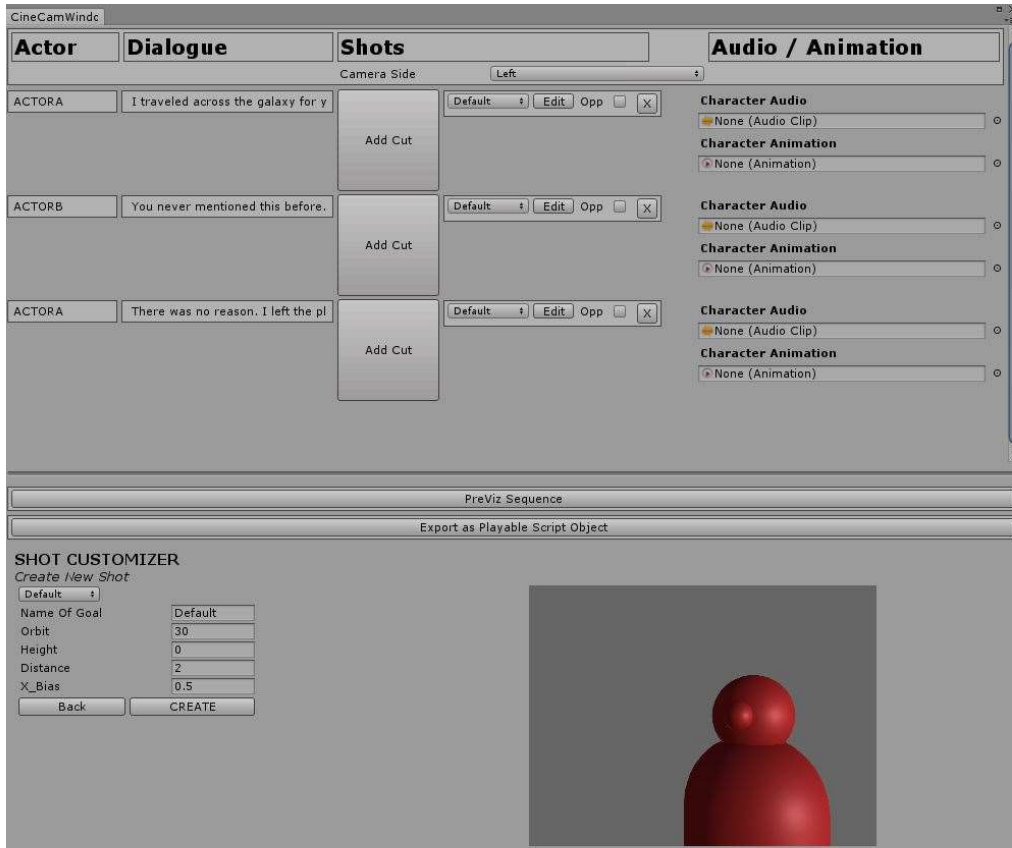

Figure 11: The complete User Interface of CineCam

3.5.1 Adding Cuts and the Opposite Action

After the Actor section and the Dialogue sections are listed, there is the Shots section. In the Shots section, the shots listed in the sequence that they will be played. The panel allows the user to add and remove shots while also setting the LOA.

The Add Cut button adds another shot to an actor's LOD. A dropdown menu lists the various shots the user can choose such as Default, Mid, Close, XtremeClose, High Angle, LowAngle, OverShoulder, and Frameshare. The edit button allows the user to edit a shot by changing the distance, height, orbit and Xbias variables associated with that shot. The *Opp* checkbox (short for opposite) sets the camera to focus on the opposite actor in the scene. For example, if the user set the Shot as "Default" on ActorA, and the *Opp* checkbox was enabled, then that shot would focus on ActorB. This feature essentially creates a reverse shot. The "X" box removes the shot from the panel and the sequence.

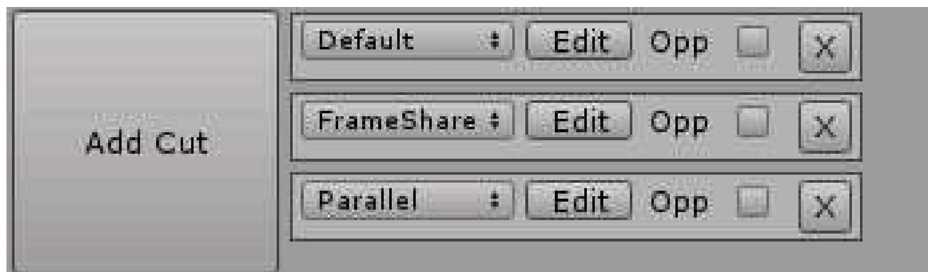

Figure 12: The Cuts Panel Interface

3.5.2 Custom Shot Creator and Editor

The Custom Shot editor is located on the bottom of the UI. It allows the user to create new shot compositions and edit existing ones. When the user clicks the Edit button the preview window displays what the shot looks like. The user can change the values in the fields, name the shot and save it as a new shot or as a replacement for the selected one. The shot creator window is displayed in Figure 11.

Every shot is defined by the orbit, height, Xbias, and distance values. When the user clicks on the "Edit" button in the UI, variables of the shot being edited are displayed in the text fields. The user can change these values and the preview window will show what the shot looks like. If a shot was edited the updated shot will be highlighted in blue in the UI. The edit window is displayed in Figure 12.

Figure 13: Custom Shot Creator

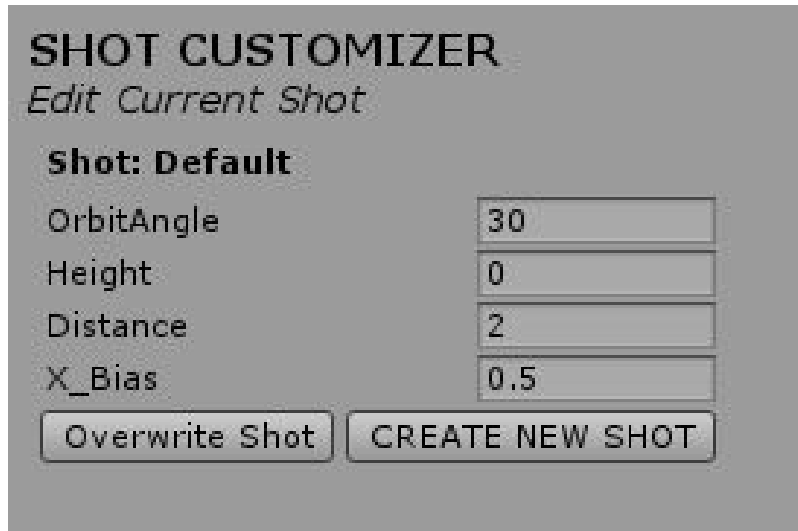

Figure 14: Custom Shot Creator

3.5.3 Dialogue Objects and Assigning Audio and Animation

With each LOD, there is an associated audio with the actor speaking and character animation. The user can assign the audio clip and animation by dragging them into the fields. The duration of the shots is informed the duration of the audio clip. For example, if the audio is 9 seconds long and there are 3 shots assigned for that LOD, each shot will be 3 seconds in duration. Basically, the shots are timed evenly within the length of the audio clip.

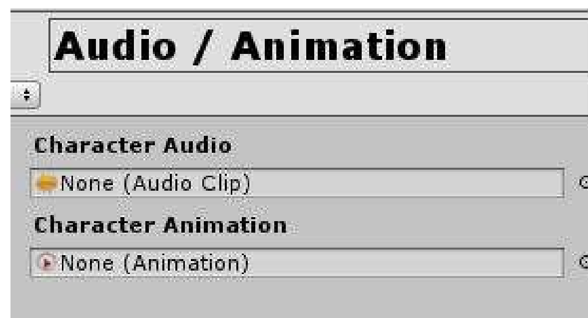

Figure 15: Adding Audio & Animation Panel

3.5.4 Exporting Shots, Dialogue Text, Audio, and Animation

After the user has finished making and assigning shots, assigning audio and animations, this data is exported into a PSO. This is done by pressing the “Export as Playable Script Object” button. The PSO stores information needed to play the sequence such as the locations of the actors, dialogue, audio clips, animation clips, and the shots. When the play function in the PSO is called from a script, the actors will be set into their respective locations when the shots were set. The game camera will play the sequence of shots along with the assigned audio and animation data. After the sequence is over the main game camera will reset itself to its original location before the sequence was played.

The “PreViz Sequence” button plays the sequence in the editor. This allows the user to experiment with different shots and visualize the resulting sequence.

Figure 16: Previz and Export Buttons

3.6 Saving Shot Patterns

The tool also allows the user to save a “shot pattern”. If a sequence of shots has been set and calculated. They can be saved as dialogue objects and applied over any audio sequence. For example, the user has saved a dialogue sequence as follows a Default, and OverShoulder shot for

dialogue for audio 1, then Frameshare for dialogue 2. The user can save the sequence as a shot pattern and apply it to any dialogue.

The user would load the shot pattern and then apply the dialogue. This has applicability because dialogue sequences often have similar patterns and they are often reused.

4. CONCLUSION & FUTURE WORK

4.1 Conclusion

CineCam's capacity to save shot idioms and apply them through a UI reduces the need to place the camera manually in 3D space. This technique has applicability not just in games, but also can be applied to previsualization tools in films as well. I realize an experienced layout/cinematographic artist who is used to navigating 3D space quickly and composing camera shots might not find this tool useful. However as mentioned earlier, this tool is designed for developers who are unfamiliar with cinematography. The built-in idioms allow them to compose cinematographically accurate dialogue sequences quickly and with relative ease.

As previously stated, in narrative games, dialogue sequences can be a large portion of gameplay. The CineCam tool facilitates a framework to create these sequences quickly. In this sense, a large portion of a narrative game is already built into the CineCam tool which can help to reduce overall development time.

4.2 Future Work

4.2.1 Extending CineCam

Future work in this tool involves adding the capability for shots such as OverShoulder and FrameShare to include 3 or more characters as it is common for dialogue scenes to include groups of actors. Currently the tool only takes-into-account 2 characters. Another feature would be having the camera follow subjects around in a scene as they are moving. Often characters can

be pacing around in the scene while they are talking. Incorporating this feature would make for a more dynamic tool.

Another future improvement would be exporting the shots, corresponding audio and animation into Unity's CineMachine's timeline feature. CineMachine's timeline is similar to an editor timeline that is seen in Adobe Premiere where the user can edit the timing of cuts and clips by scaling and sliding the cuts. Having the ability to do this would give the user further flexibility to edit the sequence.

Finally, animation clips can currently be assigned in the tool but the models do not currently play the animations at runtime. Fully integrating models to play assigned animations is subject for future work as well.

For further discussion, this tool could inform the design of immersive characters as it is important to design characters in a way that can be framed recognizably. Focus should be on the character's eyes and head as the calculations of the tool are based on that area.

4.2.2 Crowdsourcing Cinematographic Styles

In addition to extending the tool's features, it would also be helpful for researchers to be able crowdsource cinematographic styles by parametrically extracting cinematographic information from other films using the same parameters that CineCam uses to compose shots. For example, in Wes Anderson films, shots are consistently framed where the actor is centralized. This would mean there is always be low or no Xbias value to shift the camera when shots are being framed. The orbit value would be 0 or close to 0 so the character is seen head on. The distance value would be a value that would change arbitrarily depending on the user's

preference. Being able to stylize shot techniques to fit a cinematographic style would allow users to stylized their dialogue sequence to fit those of famous directors.

4.2.3 Allowing for Different Character Heights

Often in dialogue sequences characters can have different heights. The tool operates on the assumption that the characters are the same height. Future work would include the tool to take into account the different heights of the characters. It would use a character height variable to factor in the camera height. Different character heights would also change the way the OverShoulder shot is calculated as it would need to adjust to be able to include in the view a shorter or taller character.

REFERENCES

1. Cozic, L. Automated Cinematography for Games. Master's Thesis – Lansdown Centre for Electronic Arts, School of Arts, Middlesex University. Londres, Inglaterra, 2007.

2. Galvane, Quentin, et al. "Narrative-Driven Camera Control for Cinematic Replay of Computer Games." *Proceedings of the Seventh International Conference on Motion in Games - MIG '14*, 2014, doi:10.1145/2668064.2668104.

3. Lino, C., Christie, M., Ranon, R., & Bares, W. The director's lens: an intelligent assistant for virtual cinematography. In Proc. ACMMM '11. ACM (2011), 323-332.

4. Friedman, D., Feldman, Y.A.: Automated cinematic reasoning about camera behavior. Expert Syst. Appl. 30, 694–704 (2006).

5. R. Ronfard. A Review of Film Editing Techniques for Digital Games. In R. M. Y. Arnav Jhala, editor, Workshop on Intelligent Cinematography and Editing, Raleigh, United States, May 2012. ACM.

6. Lai, P., Wu, H., Sanokho, C., Christie, M. and Li, T. A Pattern-based Tool for Creating Virtual Cinematography in Interactive Storytelling. In Anonymous Smart Graphics. (August 27-29, 2014). Springer, Taiwan, 2014, 121-132.

7. Marti, Marcel, et al. "Cardinal." *Proceedings of the 2018 Conference on Human Information Interaction&Retrieval - IUI '18*, 2018, doi:10.1145/3172944.3172972.

8. P. Burelli, "Virtual cinematography in games: investigating the impact on player experience," Foundations of Digital Games, 2013.

9. Bares W. H., Lester J. C.: Cinematographic user models for automated realtime camera control in dynamic 3D environments. InProceedings of the sixth International Conference on User Modeling (UM 97) (Vien New York, 1997), Springer-Verlag, pp. 215–226.

10. Christianson, D., Anderson, S., He, L., Salesin, D., Weld, S., and Cohen, F., Declarative Camera Control for Automatic Cinematography, in the Proceedings of the Conference of the American Association for Articifical Intelligence, page 148-155, 1996.

11. Tomlinson, B., Blumberg, B., and Delphine, N. 2000. Expressive Autonomous Cinematography for Interactive Virtual Environments. In Proc. of the 4th International Conf. on Autonomous Agents.