LEVEL OF DETAILS IN FIRST-PERSON SHOOTING GAME - DH2323 Computer Graphics And Interaction

Amanda Lindqvist & Haris Vidimlic
amlindqv@kth.se; harisv@kth.se
KTH Royal Institute Of Technology

ABSTRACT

We will present an implementation of Level of Detail, LOD, that utilises the view distance and an object's velocity as a selection criterion for different LOD meshes in a first-person shooting game. Meshes with different degrees of fidelity was generated using Blender and the LOD script created in Unity which incorporate aspects of motion vectors. A minor telemetry test was performed to evaluate the LOD implementation in comparison to Unity's LOD group and no LOD. The test showed minor to no difference. Suggestions for future user studies are presented.

1. INTRODUCTION

Level of Details, also known as LODs, refer to different complexity levels of a computer-generated model or image. For example, this can be set manually when a user chooses low or high texture quality in a game. LODs are also used in real-time, most commonly when the distance to the object increases or decreases. Other uses for LODs are different velocities of objects and how they are perceived visually in regards to paracentral and peripheral vision. (Luebke et al. 2003*a*)

LODs go hand in hand with human vision, where an object's details are affected by: view distance, visual field, and the object's velocity. Therefore, it is unnecessary to render an object in full detail when it's far away, etc. The LOD of the object is instead decreased in instances where human visual perception is decreased.(Luebke et al. 2003*a*) This can greatly save on computing power and increase system performance (Velho et al. 2002), something that is crucial in complex VR-environments as they usually are demanding on the system. (Seo et al. 1999)

A well-implemented LOD should make it so that the viewer cannot see any noticeable differences between the different levels in real-time. Balancing performance and human visual perception is therefore an important aspect.

As mentioned earlier, there are different LOD criteria, where distance is the most common. Additionally, there are different LOD frameworks, such as DLOD (discrete LOD) and CLOD (continuous LOD). (Luebke et al. 2003*a*) Discrete LOD is the traditional approach to LODs, where you, during preprocessing, create multiple versions of an object with different levels of detail. Continuous LOD is when you extract the desired level of detail at run-time using live-processing. The advantage of using DLOD over CLOD is the lack of extra processing required. However, CLOD yields much smoother transitions between models. (Zach et al. 2002) We are exclusively using DLOD and will be calling it LOD throughout the report.

Unity provides a LOD technique, dubbed LOD Group, for meshes to render gameobjects with different levels of details depending on the camera's distance to the gameobject. By

predefining meshes with different degrees of geometry and the distance thresholds for which those meshes, LOD levels, are defined, you can decrease the number of GPU operations that Unity requires to render gameobjects and thus increase the performance. But as mentioned, Unity's LOD Group only depends on the view distance and does not utilise other aspects of the human visual system, such as the object's velocity, to increase system performance.

Therefore the aim of our project is to implement a LOD script that combines both view distance and the object's velocity as a selection criterion for different LOD meshes. It will be based on a first-person shooting game and created using Unity and Blender. We will present a brief evaluative comparison of the performance of our own implementation, Unity's LOD Group, and no LOD, based frames per second and provide suggestions for future perceptual user studies.

2. RELATED WORK

A previous study by Reddy (1998) presented a general model for different selection criteria of LOD based on human visual perception and evaluated their individual and combined effect on system performance.

They argue that the balancing act between human visual perception and performance greatly depends on the type of system. LOD methods based on combining velocity and eccentricity (peripheral vision) are beneficial in VR systems but only account for around 5 % of performance gain when used on a desktop display. This is because velocity and eccentricity are difficult to quantify when an individual is looking at a desktop monitor, as the head- and eye-movement aspects are ambiguous.

Furthermore, based on the displays field of view and pixel resolution, they present calculations for the highest visible spatial frequencies in relation to velocity. They conclude that for a typical desktop display an object would have to move across the screen, from the right to the left side, in at least 1.26 before a lower LOD mesh should be selected.

Another study (Parkhurst & Niebur 2004) compared and evaluated system performance gain of velocity-dependent and gaze-contingent rendering of LOD based on a target identification task on a standard desktop system. In the former, the angular velocity of the viewport in relation to an stationary environment was used and the experiment showed that velocity-based LOD had minor to no effects on the participants ability to fulfill the visual search task but that the system performance gain was evident. In general, they concluded that the nature of the task was important when balancing the trade-off between cost and benefit of LODs.

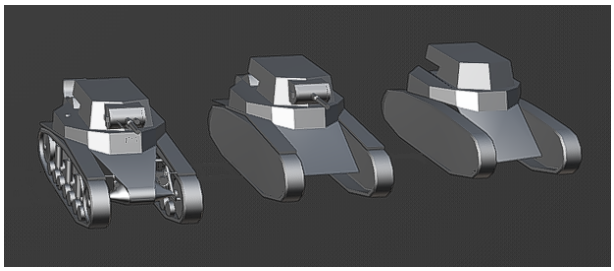
3. IMPLEMENTATION

Below we will present the implementation process of our distance- and velocity-dependent LOD. Further accounts of the process can be found in our development blog (dh2323-lodsfps.blogspot.com), which also contains a web-player version of the our game.

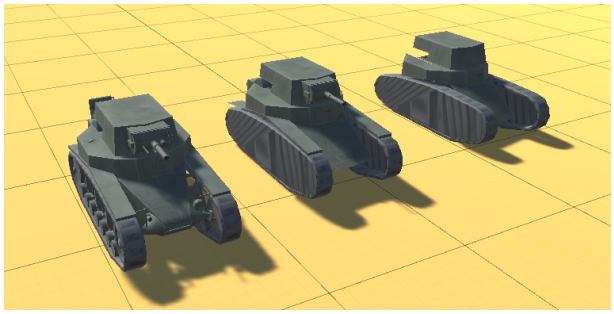
3.1. Adaptation of the first-person shooting game framework

We utilised the Tankscene from lab 3 in the animation track of this course, *DH2323 Computer Graphics and Interaction*, with additional adaptations to make it fit our scenario of a first-person shooting game. In order to implement the first-person movement we made the Main Camera a child to the tank, i.e. to the player, and added a script that tracks the mouse movement and transforms the camera view based on that. Since the mouse movement also controls the turret rotation we restricted the first person movement to the x-axis as that felt more intuitive. Furthermore, we adapted it so that each enemy randomly spawned from one of four points in order to make their movement less predictable and thus enable a more reliable testing of our own LOD implementation.

3.2. Generating LOD meshes



(a)



(b)

Fig. 1: LOD0 - left, LOD1 - middle, LOD2 - right (a) The three different LOD meshes shown in Blender (b) The three diffrent LOD meshes shown in Unity

We used Blender to generate the different LOD meshes. We chose to use our LOD implementation on the enemy tanks exclusively. Therefore, additional meshes were only generated

for them. Simply decreasing the number of triangles by a selected factor would change the overall shape too much for our liking. Therefore, we used vertex removal and edge collapsing with manual processing instead, as this would result in a model with fewer "details" yet retain its overall shape, contributing to smoother transitions between the levels. Choosing to remove vertices manually also gave us more control over the tank's components and which to then remove between each mesh. (Luebke et al. 2003*c*) However, new tank tracks for LOD1 & LOD2 had to be generated because they looked hollow after removing the wheels from LOD0. The tracks were re-modeled by adding a track "cover," as seen on LOD1 and LOD2, resulting in smoother transitions.

Model Statistics			
Model Name	Triangles	Vertices	Faces
enemy_LOD0	5,750	3,177	2,920
enemy_LOD1	1,058	697	543
enemy_LOD2	582	364	303

Table 1: Model statistics in Blender of each LOD mesh

3.3. LOD structure in Unity

The structure of our LOD implementation in Unity (see figure 2) utilises nested gameobjects and the SetActive method (Unity Documentation 2022*b*). We have an empty parent object that contains the LOD script and other common components such as the box collider and rigidbody. It has the three LOD meshes as child objects. By using SetActive(true) and SetActive(false), we control which LOD mesh is currently being displayed.

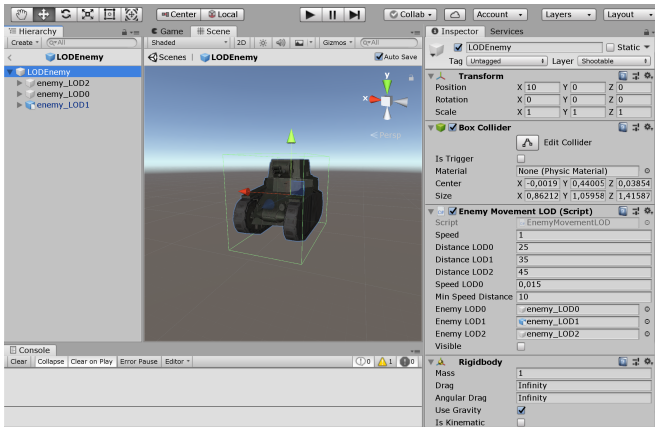


Fig. 2: Structure of LODEnemy in Unity - overall hierarchy and *Inspector* values of the parent Gameobject

Algorithm 1 Velocity and distance based LOD: *Script attached to LOD Gameobject*

```
if enemy visible then
    if timer is up then
        if length of enemyPositionlist = 2 then
            EnemyControlLOD( )
            clear enemyPositionList
        else
            EnemyViewportPosition( )        ▷ Adds current enemy pos. to enemyPositionList
        end if
    else
        timer += 1 * deltatime
    end if
end if
```

3.4. Detecting enemy's visibility

As apparent in the outlining pseudocode of our LOD algorithm (see algorithm 1), we first want to check whether the enemy is visible to the camera as to avoid unnecessary expensive operations. We attached a script to the LOD Gameobject, i.e. the enemy, that toggles a boolean *visible* variable using Unity's OnBecameVisible and OnBecameInvisible methods (Unity Documentation 2022*c*).

3.5. Implementation of object's velocity and view distance

The enemy's velocity depends on several aspects; the angular velocity of the viewport, the enemy's speed and movement, and the players speed and movement. Since the purpose of LODs is to increase system performance we wanted to minimize the use of expensive operations. Therefore, our approach was to add a timer and store the enemy's position every second. When two positions is stored we calculate the motion vector and based on its length we can derive how fast the enemy has moved across the screen under a known period of time.

We used Unity's WorldToViewportPoint to get the enemy's position, it transforms the world space position into viewport space which is normalized with values from 0 to 1. This ensures that the LOD implementation is independent of potential changes of screen size. (Unity Documentation 2022*a*)

To derive the distance, we calculate the length of the vector from the enemy position to the main camera's position, i.e. the player.

3.6. Combining and calibrating thresholds for view distance and velocity

In algorithm 2, the overall syntax of the different conditions for our implemented LOD can be viewed in pseudocode. As previously mentioned we generated three different meshes with varying degrees of fidelity. Initially, we explored the thresholds for distance- and velocity-based LOD, independently, for each of the three LOD meshes. Each of the authors evaluated thresholds on screens of different resolutions with the aim of avoiding noticeable changes, i.e. *popping*, whilst still maximizing the effects of the LODs.

Our calculation of the velocity is based on motion vectors, and how far the enemy moved in relation to the viewport under a set period of time, i.e. one second. This implementation caused there to be an effect of the enemy moving "quicker" when closer to the camera, therefore we introduced a minimum distance (*minDistance*, line 11 in algorithm 2 of where the enemy's velocity should effect the LOD. Based on (Reddy 2008) study the object needs to move across the screen in 1.26 seconds before a lower LOD should be selected. In our case that entails that the length of the enemy's motion vector should be at least approximately 0.79 (since we note the position every 1 second and the viewport has a range of 0 to 1). The value of 0.79 was used as the velocity threshold in our final implementation, but based on our own evaluation it could have been slower. Meaning the enemy could have moved a smaller distance under the same period without noticeable changes of LOD meshes.

Since the final velocity threshold requires the enemy to move so quickly in relation to the camera, it is our initial check (see variable *speed* in algorithm 2). If the velocity is less then the threshold, i.e. enemy moved a distance smaller than 0.79, we base the active LOD mesh on the distance. The decision and syntax of using three thresholds for distance is in line with (Luebke et al. 2003*b*) If, on the other hand, the velocity is above the threshold, the lowest fidelity mesh, *enemyLOD2* is active.

Algorithm 2 EnemyControlLOD() *function*

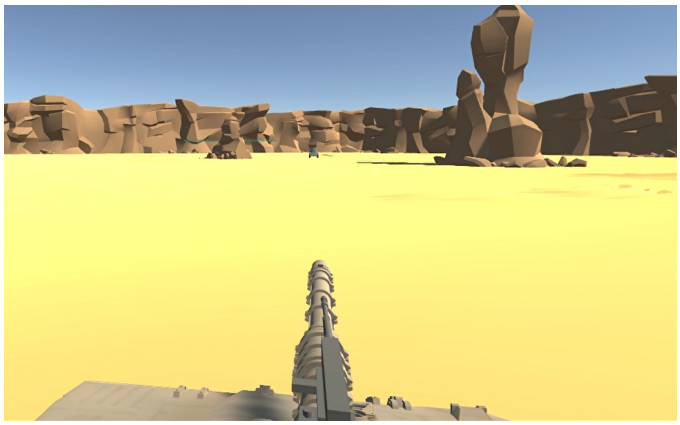
1: $distance \leftarrow$ calculate enemy distance
2: $speed \leftarrow$ calculate enemy velocity
3: **if** $speed \leq speedLOD0$ **then**
4: **if** $distance \leq distanceLOD0$ **then**
5: enemyLOD0
6: **else if** $distance \leq distanceLOD1$ **then**
7: enemyLOD1
8: **else if** $distance > distanceLOD2$ **then**
9: enemyLOD2
10: **end if**
11: **else if** $speed > speedLOD1$ and $distance > minDistance$ **then**
12: enemyLOD2
13: **end if**

4. EVALUATION - TELEMETRY TEST

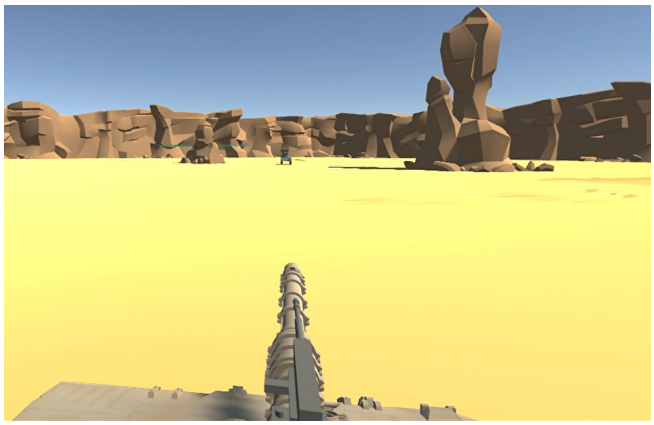
As an initial evaluation a minor telemetry test was performed where we compared three conditions: our implemented LOD, no LOD and, Unity's LOD group. The test was performed by one of the authors on a computer with 6 core i7-8750H CPU combined with a Nvidia GTX 1050Ti dedicated GPU on a 27-inch 1440p 144Hz monitor. In each of the conditions, 22 enemies where spawned and active during 5 minutes of gameplay. Unity asset *Graphy - FPS counter* was used to evaluate the telemetry data, specifically the frames per second, of each condition (Unity Asset Store - *Taxy* 2021-04-19).

5. RESULTS

The figures below show screenshots of the final LOD implementation in-game. Figure 3a illustrates the distance from player to enemy tank when LOD2 is active. Figure 3b illustrates the distance from player to enemy tank when the active LOD switches from LOD2 to LOD1. Figure 4a illustrates when the active LOD switches from LOD1 to LOD0, the highest quality mesh. Lastly, figure 4b illustrates an enemy tank with LOD2 active while the camera is quickly panned to the left. The panning speed is fast enough for LOD2 to be active, even if the tank is in the "zone" for LOD0.

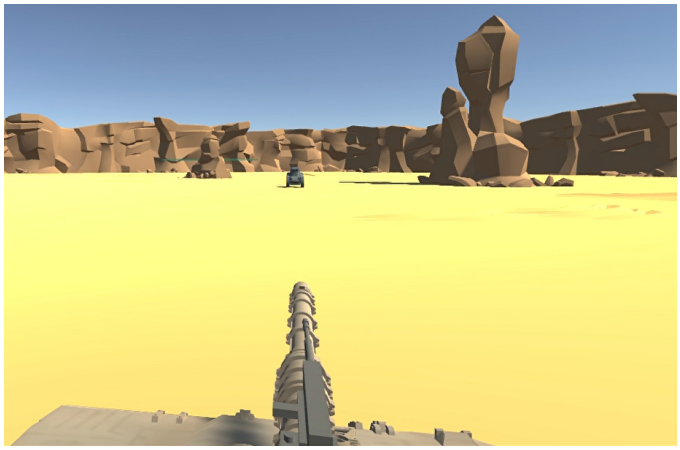


(a) LOD2 (distance)

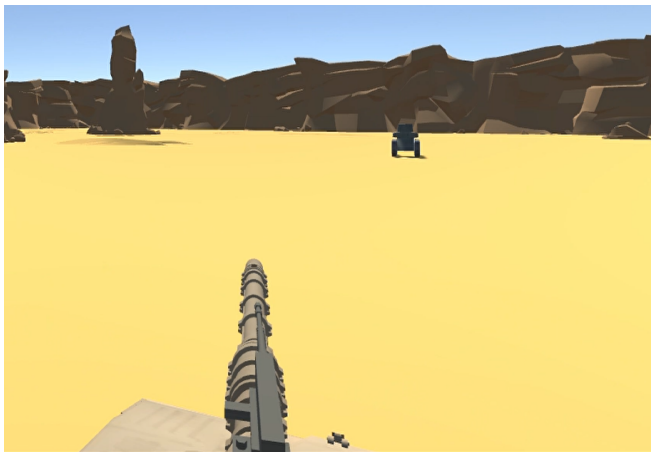


(b) LOD1 (distance)

Fig. 3: (a) & (b) Showing screenshots of different LODs during game-play and their threshold in action



(a) LOD0 (distance)



(b) LOD2 (velocity)

Fig. 4: (a) Showing a screenshot of the highest quality mesh and its threshold during game-play (b) Showing a screenshot of velocity based LOD during game-play

5.1. Results of Telemetry Test

The results of the telemetry test can be viewed in Table 2. As shown there is minor to no difference in frames per second between the three conditions.

Scene telemetry - frames per second			
Scene Name	Avg	Min	Max
TankNewLOD	137	29	144
TankNoLOD	135	30	144
TankUnityLOD	137	29	144

Table 2: Telemetry of the three test conditions after 5 minutes of gameplay and 22 spawned enemies

6. DISCUSSION

Below we will discuss the results of our minor telemetry test. Furthermore, suggestions of future work and evaluation in the form of a perceptual study will be presented.

6.1. Telemetry Test

We hypothesize that our particular scenes were not demanding enough in relation to the equipment used and therefore showed little to no difference on the telemetry data. Using LODs could potentially be more beneficial on performance when playing on a computer without a dGPU or on a smartphone. We decided on not using Unity's Profiler tool, as our scenes were very different in terms of processes which made them difficult to compare on such a close level. We therefore settled on using a simple fps monitor, which is the industry-

standard way of comparing game performance. This monitor was not made directly by Unity so we can not ensure that it is 100% accurate.

6.2. Suggestion of perceptual study

In line with the experimental setup presented in (Parkhurst & Niebur 2004), we suggest that future experiments should be performed to examine what effects, if any, the implemented LOD presented in this report has on a users ability to play a first-person shooting game. Appropriate changes to the gameplay would have to be done to create a well defined task for the participants, e.g. start with 20 enemy's and destroy as many as possible before you are reached by one. We propose a within-subject study consisting of the three previously mentioned conditions: implemented LOD, Unity's LOD group, and lastly no LOD as a control condition. The participants would not be informed of the purpose of the study and presented with the conditions in random order. Dependent variables subject to statistical analysis could be time to perform the task and shooting accuracy. These aspects would be evaluated relative to the system performance and the frame rendering time of each condition.

In the current project, thresholds for the velocity- and distance based LOD were based on the authors impressions and previous studies grounded in human visual perception (Reddy 2008). However, based on our findings less research have involved the user in the calibration of thresholds with regards to the cost benefit trade-off between frame rate and the perceived visual rendering quality. To evaluate this aspect future qualitative studies should adopt a more user-centered development process where they allow users to control and manipulate the thresholds themselves based on their perception of the ideal balance between visual rendering quality and frame rate. This should be complemented by interviews and potentially focus groups which could reveal other aspects of the users views and perception of LODs in first-person shooting games.

6.3. Future Work

An additional variable that could be implemented is an LOD based on eccentricity, peripheral vision. Adding this aspect was discussed at the start of the project and could possibly be implemented using eye tracking technology such as a Tobii eye tracker. However, limited access to this kind of equipment prevented us from combining eccentricity with velocity, which has proven to be more effective when implemented together than when implemented separately.

As this game is not particularly demanding in itself, using continuous LOD and live-processing could make the experience smoother without a big performance hit. An alternative way would be to use alpha-blending, where you give an object a range of transparency. (Dhanani & Parker 2013) For example, LOD1 could slowly be applied with decreasing transparency while LOD2 slowly fades away, making the transition between LOD2 and LOD1 smoother.

Making this game in VR would make it much more demanding as the render resolution is higher and the frame-rate needs to be high and constant for a pleasureble experience. (Seo et al. 1999) Using VR would also make the aspect of panning speed more intuitive as it would mimic the players natural movement.

References

Dhanani, S. & Parker, M. (2013), 7 - Alpha Blending, *in* S. Dhanani & M. Parker, eds, 'Digital Video Processing for Engineers', Newnes, Oxford, pp. 49–52.
URL: *https://www.sciencedirect.com/science/article/pii/B9-780124157606000076*

Luebke, D., Reddy, M., Cohen, J. D., Varshney, A., Watson, B. & Huebner, R. (2003*a*), Chapter 1 - Introduction, *in* D. Luebke, M. Reddy, J. D. Cohen, A. Varshney, B. Watson & R. Huebner, eds, 'Level of Detail for 3D Graphics', The Morgan Kaufmann Series in Computer Graphics, Morgan Kaufmann, San Francisco, pp. 3–ii.

Luebke, D., Reddy, M., Cohen, J. D., Varshney, A., Watson, B. & Huebner, R. (2003*b*), Chapter 4 - Run-Time Frameworks, *in* D. Luebke, M. Reddy, J. D. Cohen, A. Varshney, B. Watson & R. Huebner, eds, 'Level of Detail for 3D Graphics', The Morgan Kaufmann Series in Computer Graphics, Morgan Kaufmann, San Francisco, pp. 90–91.
URL: *https://www.sciencedirect.com/science/article/pii/B9-781558608382500066*

Luebke, D., Reddy, M., Cohen, J. D., Varshney, A., Watson, B. & Huebner, R. (2003*c*), Chapter 6 - Gaming Optimizations, *in* D. Luebke, M. Reddy, J. D. Cohen, A. Varshney, B. Watson & R. Huebner, eds, 'Level of Detail for 3D Graphics', The Morgan Kaufmann Series in Computer Graphics, Morgan Kaufmann, San Francisco, pp. 151–183.
URL: *https://www.sciencedirect.com/science/article/pii/B9-781558608382500008X*

Parkhurst, D. J. & Niebur, E. (2004), A feasibility test for perceptually adaptive level of detail rendering on desktop systems, *in* 'APGV '04'.

Reddy, M. (2008), 'Specification and evaluation of level of detail selection criteria', *Virtual Reality* **3**, 132–143.

Seo, J., Kim, G., Kyo, K. & Kang, K. (1999), 'Levels of detail (lod) engineering of vr objects'.

Unity Asset Store - *Taxy* (2021-04-19), '[Graphy] - Ultimate FPS Counter - Stats Monitor Debugger'. [Online; accessed 2022-05-18].
URL: *https://assetstore.unity.com/packages/tools/gui/graphy-ultimate-fps-counter-stats-monitor-debugger-105778publisher*

Unity Documentation (2022*a*), 'Camera.worldtoviewport'. [Online; accessed 2022-05-18].
URL: *https://docs.unity3d.com/ScriptReference/Camera.WorldToViewportPoint.html*

Unity Documentation (2022*b*), 'Gameobject.setactive'. [Online; accessed 2022-05-18].
URL: *https://docs.unity3d.com/ScriptReference/GameObject.SetActive.html*

Unity Documentation (2022*c*), 'Monobehaviour.onbecamevisible'. [Online; accessed 2022-05-18].
URL: *https://docs.unity3d.com/ScriptReference/MonoBehaviour.OnBecameVisible.html*

Velho, L., Cezar, P. & Carvalho, P. (2002), Mathematical optimization in graphics and vision.

Zach, C., Mantler, S. & Karner, K. (2002), Time-critical rendering of discrete and continuous levels of detail, pp. 1–8.