LpGL: Low-power Graphics Library *for* Mobile AR Headsets

Jaewon Choi
Ajou Univerisity
Department of Computer Engineering
Republic of Korea
jainersoer@ajou.ac.kr

HyeonJung Park
Ajou University
Department of Computer Engineering
Republic of Korea
jhikm1003@ajou.ac.kr

Jeongyeup Paek
Chung-Ang University
School of Computer Science and Engineering, Republic of Korea
jpaek@cau.ac.kr

Rajesh Krishna Balan
Singapore Management University
School of Information Systems
Singapore
rajesh@smu.edu.sg

JeongGil Ko
Ajou Univerisity
Department of Computer Engineering
Republic of Korea
jgko@ajou.ac.kr

ABSTRACT

We present *LpGL*, an OpenGL API compatible *Low-power Graphics Library* for energy efficient AR headset applications. We first characterize the power consumption patterns of a state of the art AR headset, Magic Leap One, and empirically show that its internal GPU is the most impactful and controllable energy consumer. Based on the preliminary studies, we design *LpGL* so that it uses the device's gaze/head orientation information and geometry data to infer user perception information, intercepts application-level graphics API calls, and employs frame rate control, mesh simplification, and culling techniques to enhance energy efficiency of AR headsets without detriment of user experience. Results from a comprehensive set of controlled in-lab experiments and an IRB-approved user study with 25 participants show that *LpGL* reduces up to ~22% of total energy usage while adding only 46μsec of latency per object with close to no loss in subjective user experience.

CCS CONCEPTS

• **Human-centered computing** → **Ubiquitous and mobile computing**; Interactive systems and tools; • **Computer systems organization** → **Embedded software**.

KEYWORDS

Augmented Reality; Energy Efficiency; Mobile Headsets

ACM Reference Format:
Jaewon Choi, HyeonJung Park, Jeongyeup Paek, Rajesh Krishna Balan, and JeongGil Ko. 2019. *LpGL*: Low-power Graphics Library *for* Mobile AR Headsets. In *The 17th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys '19), June 17–21, 2019, Seoul, Republic of Korea.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3307334.3326097

MobiSys '19, June 17–21, 2019, Seoul, Republic of Korea

ACM ISBN 978-1-4503-6661-8/19/06…$15.00
https://doi.org/10.1145/3307334.3326097

Figure 1: Study participant with Magic Leap One (a) with scenes observed from baseline (b) and *LpGL* (c).

1 INTRODUCTION

Augmented reality (AR) applications are starting to be used in various real-world application domains, ranging from industrial engineering, clinical services, field repairs, to lifestyle management [7, 13, 24, 26, 45]. To support these real-world on-site use cases, mobile untethered AR headsets such as the Microsoft HoloLens [21], Google Glass [14], and Magic Leap One [25] are now commercially available. Typically, these "untethered" mobile headsets differ from tethered devices in that they are fully self-contained and have the CPU, GPU, and networking capabilities to run full AR applications without access to more powerful computing devices nearby. Thus, they enable AR applications to be used in any location and environment, allowing a much richer set of viable use cases.

The challenge is that untethered AR headsets are *resource constrained mobile* computing platforms. In particular, they are intentionally designed to be lightweight and have minimal heat buildup. As such, untethered AR headsets have limited battery capacity (to reduce weight), and use less powerful CPU and GPUs (to reduce power usage and heat), which can limit their usefulness in many real-world use cases. For example, the Magic Leap One has only 2 to 3 hr usage time [25], which may not be sufficient for use cases that do not allow for convenient access to recharging.

A key approach to maximize battery lifetime on mobile devices involves reducing the fidelity of applications or hardware components. However, these trade-offs must be carefully managed to avoid sacrificing user experience. In this paper, we present *LpGL*, a low-power graphics library that automatically adjusts the resource usage of AR apps to reduce power consumption significantly with minimal loss of user experience. *LpGL* requires no application changes as it abstracts all energy efficiency considerations in lower layers that are invisible to user applications (Section 3). This is achieved

by re-writing, re-ordering, and selectively executing API requests in the underlying *LpGL* layer based on user perception.

Specifically, *LpGL* uses gaze/head orientation and 3D object geometry data to obtain user perception information, and combines three main techniques to reduce power consumption of applications while preserving usability;

(1) Dynamics score calculation for frame rate scaling,
(2) Mesh simplification for reducing the projected 3D object's complexity, and
(3) Culling for reducing draw call counts to minimize power consumption without impacting user experience.

These techniques operate based on the key user and mobility information provided by the AR device to enact its final power savings configuration. In particular, *LpGL* uses (a) dynamic head and eye tracking of where the user is looking at currently, and (b) the selected power usage profile (off, normal, high, etc.) to achieve an output that saves a high amount of power with minimal impact on the usability of the mobile AR application.

The primary goal of this work is to provide a solution that would work across various AR headsets – regardless of their manufacturer and software setup. As such, we focused on adapting software and device aspects only those that are accessible across all AR headset devices we surveyed (Section 2.1). From our survey, we found that most AR devices use proprietary software that provides limited access to the hardware of the AR device. In particular, the head, eye, hand, and inertial tracking subsystems of these AR devices were not easily modifiable – you could read their values easily but there was no access to change their settings to, for example, implement a duty cycling scheme. However, all the surveyed devices provided OpenGL graphics libraries that allowed good access to the rendering subsystems. Thus, to be as device independent as possible, we focused our efforts primarily on the rendering subsystems (while using inputs from the sensing components to drive our adaptation) and defer adaptation of other energy-consuming components to future work – or such a time where easy access to these subsystems becomes possible.

For this reason, *LpGL* provides an OpenGL compatible graphics API wrapper that is used to modify the application's OpenGL API calls to output a modified command set that reduces system-level power consumption. This allows *LpGL* to be highly device- and application- independent, and as transparent as possible to both applications and users. Note: users may see a different output when *LpGL* is enabled (especially if more aggressive power saving modes are selected) – however this new output is still sufficient and acceptable for the task the user is performing and minimally disturbing within the user's core focal angle. Figure 1 shows a sample scene presented using the baseline and *LpGL*.

We implemented *LpGL* on the Magic Leap One [25] (Section 4) and conducted an extensive set of in-lab experiments to evaluate its system-level performance under various configurations. In addition, we performed an IRB-approved user study with 25 student participants to understand the usability impact of *LpGL* (Section 5). Our results from in-lab experiments and the user study show that *LpGL* reduces the power usage by as much as ~22% with only marginal (~46μsec) added latency per displayed object on a frame and minimal loss in subjective user experience levels. This power reduction translates to achieving 3.9 hours of continuous system lifetime, compared to 3.0 hours when using the baseline graphics library directly. Furthermore, our experiments show that *LpGL* results in similar or lower device temperature with the baseline approach while achieving higher frame rates.

The major contributions of this paper are as follows:

- To the best of our knowledge, this work presents the first detailed power measurement of the Magic Leap One, a commercial AR device, to inform and guide the design of *LpGL*.
- Combining mobility features with rendering techniques to allow *LpGL* to reduce energy costs from GPU and memory usage in a scalable and generally applicable manner.
- Detailed in-lab power and device temperature measurements showing the efficacy of *LpGL*.
- Results from a 25 participant user study showing that *LpGL* does not impact usability, accuracy, nor task times.

The remainder of this paper is structured as follows. In Section 2, we show our preliminary studies on the power consumption measurements of the Magic Leap One AR platform. Next, in Section 3, we present *LpGL* and its system architecture with design considerations based on the results obtained from the preliminary study. In Sections 4 and 5, we validate our system's efficacy in terms of performance measures (e.g. power consumption, latency and heat) and user satisfaction levels in controlled setting and through a user study, respectively. Section 6 deals with opportunities, limitations and future directions of *LpGL*. Finally, we introduce related work in Section 7 and conclude the paper in Section 8.

2 PRELIMINARY STUDY: AR HEADSETS

2.1 Background

In this work, we are developing power management solutions for untethered mobile AR headsets, and base our approach on two core characteristics of these types of headsets - (1) the variable scene sparsity, and (2) the ever changing user perception. A key difference of dedicated AR headsets is that, unlike virtual reality (VR) applications and mobile phone-based AR applications, AR headsets *do not need to render backgrounds*. In particular, VR or phone applications require the system to draw a virtual *background*, and mobile AR applications need to overlay images captured from the video camera on the screen. However for AR headsets, the user directly observes "reality" through a translucent lens, and only the virtual objects (e.g. 3D models) are augmented on the scene with an empty background. As a result, the problem scope of energy efficient object rendering is different from full-screen rendering systems. In addition, unlike tethered devices, untethered headsets have to perform the computationally expensive rendering process *on the mobile device itself* rather than on a nearby PC and simply displays the content on the display.

Another key characteristic of mobile AR headsets is that they require some form of orientation or *user perception* information. For example, devices such as the Hololens exploit gyroscope-based head orientation data and the Magic Leap One offers both gyroscope and gaze tracking data. These sensors inform the device on what direction the user's visual attention is focused towards and what

they are paying attention to. This is important as the device needs to understand where to focus its limited computational power on.

Finally, as stated in the introduction, most commercial untethered AR headsets are currently being released as "closed" platforms where a large amount of the software and sensors running the device is not accessible to third-party developers, making it difficult to access low level information and optimize system performance. For example, the Magic Leap One uses its Lumin OS [25], which is a sandboxed platform that provides limited access to low level information – thus limiting what third-party programs can do for performance enhancement. This is similar to the HoloLens [21] which uses the Universal Windows Platform (UWP) [37]. Naturally, this complicates the design and implementation of optimization approaches for most (even experienced) developers and suggests the need for an underlying layer to optimize the energy usage silently and obliviously to the application. Fortunately, in all cases, we found that AR devices all supported OpenGL. Thus, we chose to implement our performance improvements in the OpenGL layer as way of achieving cross-device compatibility – albeit at the loss of access to sensors and information, such as the eye and gaze tracking hardware, not accessible through graphics library APIs.

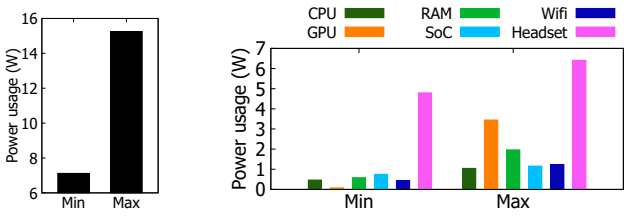
2.2 Detailed Power Measurements

We next performed a detailed study to understand the power usage characteristics of the Magic Leap One commercial untethered AR headset. This is a representative of typical untethered AR headsets and consists of three core components: computation, display, and networking. Our goal was to identify the power consumption of each component under various workloads and use these results to focus our power conservation solutions.

The Magic Leap One consists of two major (portable) components: the headset and the computation device. The headset contains a liquid crystal on silicon (LCoS) display, offering a fixed 1280x960 resolution, with a gaze tracking device and IMU sensors. The computational device, which is wired to the headset, contains an Nvidia Tegra X2 SoC with two Denver 2.0 64-bit cores and four ARM Cortex A57 64-bit core. It also contains an integrated Pascal-based GPU with 256 CUDA cores, with 8 GBs of RAM. Finally, the computation device also contains a Murata 802.11 ac/b/g/n and Bluetooth 4.0 chipset. We could not find any detailed breakdowns of recently-released untethered AR headsets' power consumption. Thus, this preliminary study is also a good reference for other researchers wanting to optimize power consumption of AR devices. To perform this preliminary study, we ran tasks that used different system configurations (e.g., object complexity, display configurations, and network data rate), and measured the power usage of each component using the Magic Leap One developer interface. Unless explicitly specified, the frame rate was set to 60 frames per second (fps), the default for Magic Leap One. Prior to our studies, we validated the accuracy of the Magic Leap One power measurement tool to confirm that its measurements match the device's actual operation lifetime.

2.3 Minimum and Maximum Power Usage

We first performed a simple experiment to measure the minimum and maximum power consumption on the Magic Leap One. For



(a) Total power consumption

(b) Per-component power consumption

Figure 2: Power usage breakdown for minimum and maximum performance cases on the Magic Leap One.

the minimum case, after switching on the device, we displayed no content, the controller was not connected to the headset, and all computing and networking features were disabled except for the always-on baseline functionality. For measuring the maximum power usage, we set the device to render at maximum rate (1.0M polygons at 60 FPS), send packets at full line rate (~14 MB/s over UDP), and connected the controller to the headset via Bluetooth. Figure 2 plots detailed power measurements for the two extreme cases. We observe that at the maximum, the power consumption increased by more than 110% compared to the minimum case. Furthermore, we notice that the headset, which includes sensors for gaze tracking, gyroscope, and the LCoS display, dominates the power usage. Even though the headset is consuming nearly 42% of the total power usage when operating at maximum, unfortunately, like many commercially available AR headsets, the Magic Leap One does not allow developers to control headset sensors' parameters (such as enforcing an energy-optimized duty cycling).

Fortunately, even without the headset, the computational units including all the processors and networking interfaces still consumes, compared to minimum case, as much as 6.5 W more and ~58% of the total system power. Thus, optimizing the power consumption of the computational unit can still result in significant power savings and increased usage time of the AR device. For example, we found that the Magic Leap One, using its 36.7 Wh battery, could only run for about 2.4 hours at the maximum power usage configuration and for 5.1 hours with the minimum configuration.

2.4 Graphics Rendering Components

We first look into the impact the graphics components of the Magic Leap One has on the overall power usage. A typical "graphics pipeline", in this case for AR devices, consists of computationally heavy operations (e.g., matrix computation and rasterization) that use source information representing 3D objects (e.g. triangle and texture information) and output rendered 2D images. For seamless user experience, the rendering pipeline should run at 60 fps or higher. However, this high fps requires a large amount of computation, which in turn requires a lot of power usage.

Before measuring the power consumption of the graphics pipeline, we first explain how applications interact with the graphics rendering libraries. To render a specific geometry (e.g., 3D object), a typical application will need to send three types of information to the graphics API: (1) vertices and their topology information, (2)

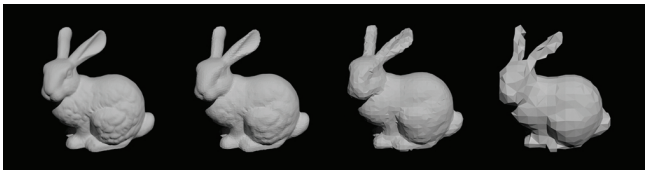
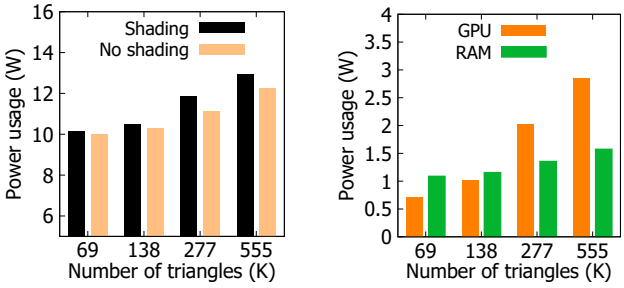
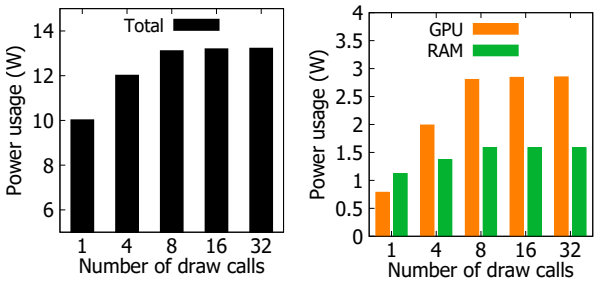


Figure 3: The Stanford Bunny [44] in four different resolutions, with different number of triangles.



(a) Total consumption (b) GPU and memory with shading

Figure 4: Power consumption plots for varying object complexity (object triangle count).



(a) Total consumption (b) Power breakdown

Figure 5: Power consumption for varying number of draw calls (e.g., objects) at 60 fps.

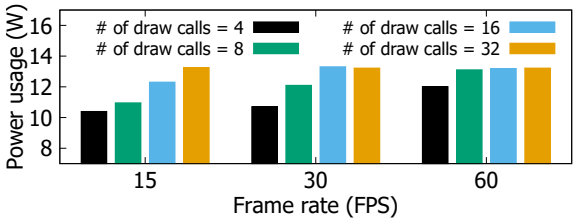


Figure 6: Power consumption for rendering the 69K-triangles Stanford Bunny with varying frame rates.

transformation matrix of a model, and (3) command to draw the object (i.e., draw call).

Number of vertices: To quantify the power consumption for different *number of vertices*, we use the widely used Stanford Bunny model (Figure 3) [44] in four different resolutions. The left-most bunny in Figure 3 is the full resolution object containing 69K triangles, with the subsequent bunnies containing 16K, 3.7K, and 900 triangles, respectively. Note: a triangle is a basic mesh object, comprised of multiple vertices, that is used to create the final 3D object – the more triangles used, the more realistic and smoother the final object will appear to the users.

The power consumption results in Figure 4(a) for rendering these four bunnies suggests that the Magic Leap One consumes a fixed amount of power (~10 W) to render up to 138K triangles. Beyond this point, the power consumption starts to increase. We also observe that the shading process consumes a significant amount of power. Quantitatively, when increasing the object complexity from 69K to 555K triangles, we observed a 3.6 W (35%) increase with shading enabled, and a 1.8 W (17%) increase without shading. From Figure 4(b), which plots the GPU and memory usage for different object complexities with shading (the most impactful components in this setting), we observed that the power consumption increase is caused by the additional GPU and memory usage – with the GPU, by itself, consuming nearly 4x more power as the complexity increases. These results suggest that controlling the number of vertices in the 3D object model (to reduce image complexity) can be an effective strategy in lowering the device's power usage.

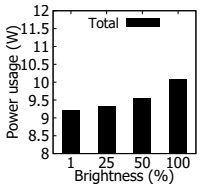
Number of draw calls: Next, we examined the case where *multiple* objects are drawn on a single scene – i.e., when multiple *draw calls* are issued. To test this, we issued up to 32 draw calls using a Stanford Bunny with 3.7K triangles. Figure 5 presents the power consumption for this test. We observed that increasing the number of draw calls from 1 to 32 resulted in a ~31% increase in power. From Figure 5(b), we observed that this increase is (mostly) due to the GPU and memory power usage. Note: when the number of draw calls reached 16 and 32 (16 and 32 bunnies on the screen), the 60 fps target could no longer be achieved (44 and 24 fps achieved, respectively) and we observed a cap on the total power consumption.

Screen complexity and frame rate: Figures 4 and 5 together provide us hints on how power usage can be effectively reduced on untethered AR headsets. As shown previously, a scene that requires many triangles, either due to a single complex object or many simpler objects, will result in high power usage. This suggests that controlling the *frame rate* can impact the system's power efficiency. In particular, high frame rates will require more pipeline iterations and draw calls within a given time interval, which will heavily impact the system's power consumption. This is shown in Figure 6 where we display 16K-triangle Stanford Bunnies using 15, 30, and 60 fps with 4, 8, 16, and 32 draw calls. We observed that as the overall complexity of the scene increases, the total power usage increases and saturates at ~13 W.

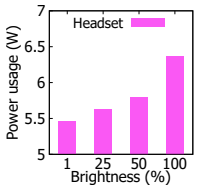
2.5 Display

A major power consuming component on mobile devices is considered to be the display [4, 8]. The Magic Leap One uses LCoS displays, which could have very different power usage patterns from prior work studying LCD-based phone displays [23] and OLED-based displays [47].

An LCoS display is a micro-display that uses a ferroelectric liquid crystal layer, containing individual electrodes, on top of a silicon backplane. A CMOS chip controlling the electrode voltages is installed below the chip surface. A common base voltage for

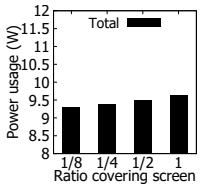


(a) Total power

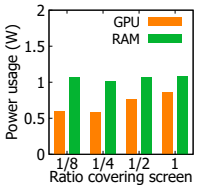


(b) Headset power

Figure 7: Impact of screen brightness on power consumption

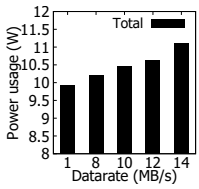


(a) Total power

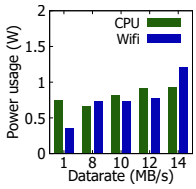


(b) GPU/memory power

Figure 8: Impact of object size on power consumption



(a) Total power



(b) WiFi module power

Figure 9: Impact of network data rate on power consumption

all the electrodes is supplied by a glass cover sitting on top. Such display technologies are also used in the Microsoft Hololens and the Google Glass [29].

To understand the power consumption of the display, we performed two experiments where we changed the (1) *display brightness* and (2) *object size*. To study the impact of brightness, we configured the Magic Leap One to show a pure white full screen image with different brightness. Figure 7 plots the power usage for brightness levels from 1-100%.

We observed that the headset's power consumption increases as the brightness increases. The difference in power usage between the lowest and highest brightness levels was ~0.9 W (9% increase). This suggests that dynamically adjusting brightness can potentially improve the system lifetime [27]. However, currently for the Magic Leap One, the brightness setting can only be configured manually by the user and cannot be adjusted automatically by applications or on a per-object basis. Thus, while schemes to reduce brightness can improve the power consumption, we cannot use them in our specific implementation for Magic Leap One.

Figure 8 plots the power consumption when displaying different-sized solid squares on Magic Leap One at maximum brightness level. We observed that as object size increases from 1/8 of the screen to 100%, there was a ~7% increase in total power usage. Most of this was from increased power consumption by the GPU and memory components due to larger objects being rendered on the screen. We chose not to dynamically reduce the object sizes in our implementation, even though it could save power, as our studies revealed that users would quickly notice size differences between objects.

2.6 Wireless Networking

Finally, the wireless networking component is often considered to be a major power consumer in mobile systems with a number of solutions proposed to reduce its energy/power consumption [3, 31, 40]. To investigate the impact of the networking module in mobile AR headsets, we connected the Magic Leap One to a nearby WiFi AP and transmitted packets at data rates of 1MB/s, 8MB/s, 10MB/s, 12MB/s and 14 MB/s, respectively, over UDP, to a nearby sync server. Note that all other computational features were turned off, only with an empty white screen on the display.

The results in Figure 9, indicate that power consumption increases by ~1 W as the data rate increases from 1MB/s to 14MB/s, and most of this increase is due to the CPU and WiFi modules. In this paper, we do not provide any schemes to optimize the power consumption of the wireless networking component for two main reasons; (1) there are already many proposed techniques [1, 9, 38, 39] that we could reuse, and (2) more importantly, our survey of AR applications revealed that very few used the networking component in any significant way. Thus we focused our efforts on reducing the energy consumption of always-used components instead.

2.7 Summary

Our preliminary study using the Magic Leap One suggests that a 3D object's geometric attributes, such as the object complexity and the frame complexity, along with the frame rate of the application, are crucial (and controllable) factors in modifying the energy usage of untethered AR headsets. In the next sections, we show how we design and evaluate a solution that uses these insights to effectively increase the battery lifetime of these devices.

3 LOW-POWER GRAPHICS LIBRARY

The results from our preliminary studies suggest that a combination of various geometric attributes (e.g. number of triangles and draw calls) and system-level parameters (e.g., frame rate) need to be considered when optimizing the AR application's graphics rendering requests for power efficiency. Moreover, if there are several 3D objects to be displayed, each object may need to be optimized separately depending on their texture or motion characteristics. Managing this individually for each application may be a significant burden and learning curve for the developers. Therefore, having a dedicated library implementation that manages object rendering, while considering various power-affecting factors, can be beneficial for mobile AR application development. Such operations should be kept transparent to the application, the added computation should be minimal, and should take in consideration the user perceived object quality. To this end, we summarize the design goals of a low-power graphics library for mobile AR headsets as follows:

- Maintain a transparent pass-through layer between the native graphics library and the application for various programs to easily inter-connect without putting the responsibility on the application developer.
- Light-weight and computationally tractable schemes are needed to assure that power and resource usage of the implementation does not outweigh its power savings.
- Manage 3D object display based on how the user perceives (both the physical visibility and quality) the AR environment. While achieving power-savings is attractive, maintaining application quality is still important.

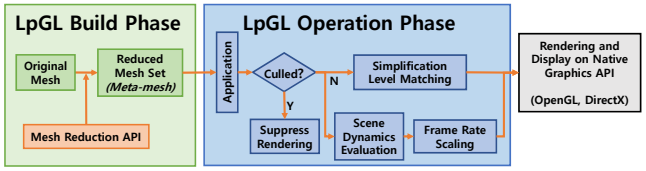


Figure 10: *LpGL* – the architecture and its functions. The native graphics APIs of *LpGL* is implemented on OpenGL and DirectX 11.

With these goals in mind, we design *LpGL*, a Low-power Graphics Library for mobile AR headset applications. In the software stack, *LpGL* is positioned between the application and the system's native graphics library (e.g., OpenGL ES, DirectX) (Figure 10). To the application, *LpGL* provides OpenGL compatible front-end APIs with OpenGL ES and DirectX implementations as a back-end native graphics API. For an application that uses these APIs, *LpGL* intercepts the graphics-related calls to understand what the application targets to draw (e.g., type/location of drawing, quantity of models). In addition, *LpGL* obtains user view points from the headset to gather user perception physics (e.g. angle of sight, gaze). Then, these information are combined to optimize the drawing process with respect to other system parameters, such as the frame rate and geometric complexity, so that power usage is reduced while minimizing user perceived quality loss. The following subsections provide details on the *LpGL* design.

3.1 *LpGL* Front-end APIs

Front-end APIs of *LpGL* are designed to be compatible with the OpenGL APIs [17]. This results in no changes to the application's code and allows *LpGL* to easily intercept and capture the contents to be drawn on the display. Through the APIs, *LpGL* hooks to the graphics-related calls from the application and passes appropriate information to its sub-modules for further processing, prior to sending commands to the system's native graphic library.

To achieve this transparency, we design our software so that all *LpGL* functionality are enabled through compiler configurations. The application developer simply makes changes to the compiler configurations to use *LpGL*, and at compile time, the source code binary and necessary 3D objects are modified to support the added features of *LpGL*. These, compile time configurations allow *LpGL* to intercept all graphics-related calls prior to entering the native graphics stack. Specifically, through this process we (1) initialize *LpGL*, (2) configure a command queue in which application API calls can be gathered before being pushed to the system graphics stack, and (3) load a mesh with different complexities to perform mesh simplification as we will detail in Section 3.2.

OpenGL-compatible front-end APIs supported in *LpGL* consists of three parts: APIs for initializing the drawing data, transforming the geometric objects, and APIs for the actual drawing. The following presents a subset of the APIs that *LpGL* supports.

Initialization APIs: The initialization APIs contain information on what objects will be drawn, including vertices and topology information (e.g. `glBufferData()`, `glVertexAttribPointer()`). These APIs allocate space on the GPU Video RAM (VRAM) and store graphics-related data such such as geometry, image or transformations on the reserved buffers.

Transformation APIs: Transformation APIs apply geometric transformation (scaling, rotation and translation) to the target object using call such as `glScale3f()`, `glRotate3f()` and `glTranslate3f()`. Furthermore, transformations in *LpGL* can be composed with matrix stacking APIs as well (e.g. `glPushMatrix()`, `glPopMatrix()`).

Draw call APIs: Draw call APIs send commands to draw objects to the GPU by indicating the geometry buffer and passing a flag to notate the objects' topological structures such as points, lines or triangles (e.g. `glDrawArrays()`, `glDrawElements()`).

Our current *LpGL* implementation provides essential APIs for "fixed pipeline" rendering. We are aware that more recent graphics programming uses the "shader". We plan to support this in our next design of *LpGL* and position this work to validate the effectiveness of different graphics pipeline optimization technologies with respect to power-efficiency.

3.2 Mesh Simplification

As we saw in the preliminary studies, the number of triangles that consist a 3D object (i.e., object complexity) heavily impacts the power usage of a mobile AR application. However, when displaying multiple complex objects in a single scene, often the user's perception (e.g., direction of sight or gaze) is towards only a subset of the objects. Therefore, it makes sense to simplify objects that are out of the user's focal angle to minimize the power used for processing these objects.

To do so, we start by identifying the number of triangles that an application intends to draw using the `glBufferData()` call and the `glVertexAttribPointer()` call in the OpenGL APIs, which provides vertices and topology information.

Next, to reduce the number of triangles that consist an object while maintaining visible quality, we apply *mesh simplification*. Mesh simplification is used in various graphics applications to minimize computation costs for less-focused objects. Among various methods, we used Audodesk Maya's Mesh Reduction API [33].

However, mesh simplification can be computationally heavy when performed on the mobile headset itself. Therefore, all potentially displayed objects in *LpGL* are simplified *in compile time* (on the development PC once) to create two additional versions of simplified meshes. Here, one version is a "less" simplified version (level 1 simplification) and the other is heavily simplified (level 2). For simple memory loading, the geometries of the original, level 1 and level 2 simplification meshes are combined to form a "meta-mesh" of an object. We note that some applications dynamically load 3D objects in run-time. To reduce object complexity for those not simplified in compile time, *LpGL* also includes a light-weight mesh simplification scheme based on Quadric Error Metrics [12] to reduce mesh complexities in run-time.

In operation, the meta-mesh of the target object is loaded to the memory upon a display request. As soon as a decision is made on what version of the object should be displayed, the meta-mesh is split, and only the geometry for the target complexity mesh is passed to the GPU.

In determining when and what version of the object will be displayed, we utilize gaze tracking or head orientation information

provided by the mobile headset. Based on geometric information on how far an object is away from the user's focal point, we divide the user's field of view (FOV) in three: core focal angle, near peripheral sight, and far peripheral sight. Assuming that the user's core focal angle is within ϵ° from the current focal point extracted from the gaze tracker or head orientation information, objects beyond this angle distance can be simplified. If the object is located 1-3×ϵ° away from the focal point, we present the level 1 simplified mesh, and use level 2 for all objects farther away. Based on findings from previous literature [41], we set ϵ° to 10° and validate this using experiments in Section 4.

3.3 Frame Rate Control

Our preliminary results show that frame rate has significant impact on the system's overall power usage. However, frame rate control should be done with care, since low frame rates may drastically drop the user perceived application/scene quality. Frame rate control in *LpGL* is designed based on the intuition that frame rate need not be constant across all visual contents [30]. It should be kept high to maintain the original quality for contents that change frequently (e.g., high level of scene dynamics), but can be lowered without loss of user-perceived quality when contents are less dynamic. To exploit this idea, *LpGL* implements a *Scene Dynamics Scoring Module*, which determines and quantifies the dynamics of frame contents using the scene's geometric information. Based on this score, the *Reactive Display Complexity Control Engine* (Section 3.5) determines an appropriate frame rate.

For example, if we define a scene's dynamics as the difference between subsequent frames, image-based schemes that use full contextual information, such as the structural similarity (SSIM) or peak signal-to-noise ratio (PSNR) can be used [46, 48]. However, they require significant amount of computation to calculate features from the entire screen, and thus unsuitable for resource limited mobile devices [23, 46]. Furthermore, with these schemes, it is difficult to compute scene dynamics until the frame is fully rendered for object rendering applications. We find this and the memory copy from the GPU's frame buffer as computational waste.

Moreover, image-based metrics may not be sensitive enough for small portions of changes in the scene. Take a case in which a small object such as a bullet passes through the user's scene. The bullet will travel fast, but if we compute the SSIM/PSNR for the entire frame, the contextual changes will only be marginal. Such schemes can suggest that there is only a small level of dynamics and decrease the frame rate accordingly. In applications where drawing is done on an object-basis, we need a scheme more centered towards the objects themselves, rather than the entire frame image.

Such observations led us to propose and design a *Distance-based frame dynamics scoring module*, an *object geometry-based* light-weight approach for determining scene dynamics. In our approach, the scene dynamics is defined as the maximum distance change of objects within two consecutive frames computed over all objects in the scene. Note that this distance is defined on the user's view point perspective. Therefore, even when the object is stationary and the user's view point changes (e.g., motions with the mobile AR device), the dynamics can be captured. To minimize computation costs, instead of using the original detailed coordinates of the 3D object, we set a "bounding-box", which is a box that best fits all the coordinates of the original object, and only compute the distance based on the box's center point. Such an approach allows *LpGL* to compute the dynamics level of the scene before the scene is rendered, to suppress any unnecessary rendering and memory copy operations.

3.4 Culling

In addition to frame rate, the number of draw calls also impact the power consumption of a mobile AR headset. Issuing draw calls are essential for drawing objects to the display, but executing draw calls for objects that cannot be physically seen (e.g., object out of current scene boundaries) would be a waste. Most widely used graphics pipelines already exploit "view frustum culling" to account for such cases [2, 5, 22]. Specifically, the culling process determines whether or not there is a chance for the target object to be observed and suppresses the drawing of the object if not.

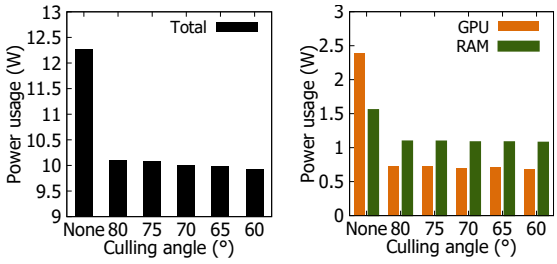
While culling itself is already widely used, we've identified a point in which the main philosophy of "not processing unseen objects" can be further exploited to reduce power consumption even more. In currently used view frustum culling, when a user is observing an object in the x° field, objects in the $x + 180^\circ$ range will still be processed and pay the computation cost up to the point where the object geometries are known to the graphics library after being fully transformed to the 3D space. However, objects beyond peripheral vision can not be seen, and we propose to eliminate *all* the waste in the culling process.

Again, we exploit the user perception data, geometry information of all objects, and apply the bounding box-based approach. By doing so, instead of transforming all vertices to the 3D space, we can selectively transform just the eight corner points that consist the bounding box. We infer the user's position in the 360° space, and use it to compute the angular distance τ° between the user's view point direction vector and the vector of the object bounding box's center position. Then, given a typical user's maximum FOV, if τ° is not included in the FOV, we determine that there is no chance for the target object to be seen in the current scene. Using this simple method, *LpGL* improves view frustum culling and suppresses the entire graphics pipeline process.

3.5 Reactive Display Complexity Control

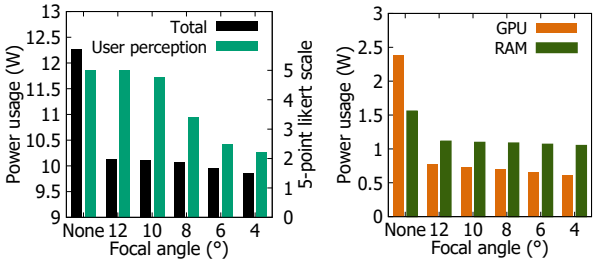
The mesh simplification, scene dynamics scoring, and culling techniques discussed until now are designed based on the idea that we can utilize the relationship between the user's view point (from the gaze tracker) and the object's geometry. To effectively combine all *LpGL* features, we design the Reactive Display Complexity Control (RDCC) engine. The RDCC engine offers a framework that exploits the three core features of *LpGL* with minimal computational overhead, and provides a fine-grained display control environment. Through RDCC, *LpGL* reactively controls graphics-related parameters based on the user's perception of the scene.

A key idea of the RDCC engine in *LpGL* is to focus the computation on only the bounding-box surrounding the geometry rather than the full geometric information. Take the culling process for example, which is interested in identifying whether the target object is within the user perceived scene or not. Given that the bounding



(a) Total power usage (b) GPU/memory power usage

Figure 11: Power consumption for enabling *LpGL*'s culling at different angles.



(a) Total power usage and 5-point Likert-scale score on scene quality (b) GPU/memory usage

Figure 12: Power consumption and scene quality perception scores for varying core focal angles.

box provides the minimum and maximum points of the object, if we simply identify that the bounding box is outside the user's sight, we can easily remove this object from the graphics pipeline. Similarly for frame rate control and mesh simplification, the RDCC engine makes its decisions based on features (e.g., scene dynamics or object position) extracted from the bounding box.

Overall, RDCC operates as follows. For each object draw call, the culling process first takes place after identifying the object's bounding box. Then a mesh of a proper detail (e.g., original object or simplified mesh) is presented after determining that the object is currently within the core focal angle or near the peripheral angle. Finally, the target frame rate is configured based on the objects' dynamics in the scene to project them through the native graphics library's drawing APIs.

4 IN-LAB EXPERIMENTS

We start our evaluations of *LpGL* using controlled in-lab experiments to analyze and understand the performance of *LpGL* under different experimental settings. Specifically, we measure the power usage for different culling angles, focal angles, and motion dynamics to capture the impact of each *LpGL* feature, (1) culling, (2) mesh simplification, and (3) screen dynamics-based frame rate control, respectively. We also gathered qualitative measurements using a small-scale user study of five participants to understand how parameter changes in the three criteria affects the perceived object quality to determine the appropriate values for our application-oriented user studies (discussed in Section 5).

4.1 Culling Angle

To identify the optimal culling angle for the Magic Leap One, we first built an application to display 72 small bunnies horizontally with the center bunny (right in front of the user's field of view) colored in green and the other 71 colored in white, arranged evenly in a 360° circle around the participant – with each bunny separated by its neighbor by $360/72 = 5°$. During the experiment, we asked each participant to focus their attention on the green bunny and inform us when they noticed something changing in the scene. We then slowly removed bunnies from the scene starting with the leftmost and rightmost bunnies (they were always removed in matching pairs) until the participant noticed something changing in their visible scene.

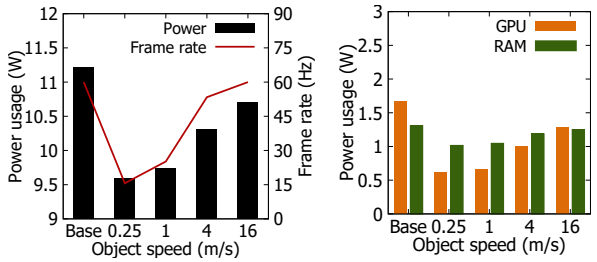
This experiment was designed to identify the visual angle at which participants could notice changes to the scene with their peripheral vision. This angle is important to know as objects outside this noticeable area can be culled by *LpGL* (i.e. not rendered) to save power. On the Magic Leap One, the effective maximum field of view is 80° – thus we started our tests from 80°, as anything larger would not be visible anyway (as the Magic Leap One would not display it).

Figure 11 shows the reduction in power consumption as the culling angle moves from "None" to 80° and lower. We observe that, depending on the scene complexity, significant amounts of power (20% or more) can be saved by not rendering objects that cannot be seen by the user. However, this culling has to be balanced with usability. From our study we noticed that, due to the small field of view of the Magic Leap One, any culling in the visible area was immediately noticed by the participants. Thus, despite increased power savings potential, to maintain the user experience, we set the culling angle for the Magic Leap One to 80° (to match its small field of view) in all future experiments.

4.2 Focal Angle

LpGL defines two types of focal angles: *peripheral focal angle* and *core focal angle*. For objects that are farther away than the peripheral focal angle, a very simple mesh is presented (level 2 simplification), and for objects within the *core focal angle* the highest quality mesh is presented. If the object is located between these two angles, a level 1 simplified mesh is displayed. We set the peripheral focal angle to 60° based on findings from previous literature [6, 16], and used the culling angle application to understand the impact of changing the core focal angle on the power and user perception. For this experiment, we placed the bunnies 2° apart to observe the effects at finer granularity.

During the study, we randomly set the core focal angle to different values and asked the participants to focus on the green bunny in the center and rank the quality of the scene using a 5-point Likert scale (very low: 1, very good: 5). With decreasing core focal angles, the bunnies that are located close to the user-focused green bunny will start to be presented as a simplified mesh. Our results, shown in Figure 12 for both power consumption and user perceived quality, suggest that the participants started to quickly notice changes when the core focal angle was smaller than 10°. Again, similar to



(a) Total power usage and achieved frame rate for varying object speeds

(b) GPU/memory usage

Figure 13: Power consumption and scene quality perception score for varying object speeds.

the culling angle, despite the power benefits of a narrow core focal angle, we set the focal angle to 10° to minimize the usability impact.

4.3 Object Speed and Frame Rate

Next, to investigate the impact of *LpGL*'s dynamic frame rate control, we conducted an experiment where four 69K triangle bunnies are moved up and down the scene. We varied the speed of the bunnies from 0.25-16 m/s keeping each bunny 5 meters away from the user. At 0.25 m/s (low dynamics), we expect *LpGL* to use a low frame rate (15 fps), while at 16 m/s we expect *LpGL* to use the maximum frame rate (60 fps). *LpGL* uses three levels of frame rates, 15, 30 and 60 fps, and we configured it based on our initial user tests to move from 15 to 30 fps if, across two frames, at least one object moves across more than 15% of the entire scene. If an object moves across more than 30% over two subsequent frames, we set the frame rate to 60 fps. From the results shown in Figure 13, we note that even at 60 fps displaying the same objects, *LpGL*'s power consumption is lower compared to the baseline (due to culling, focal angle etc.) and it's consumption rate is much lower at lower fps.

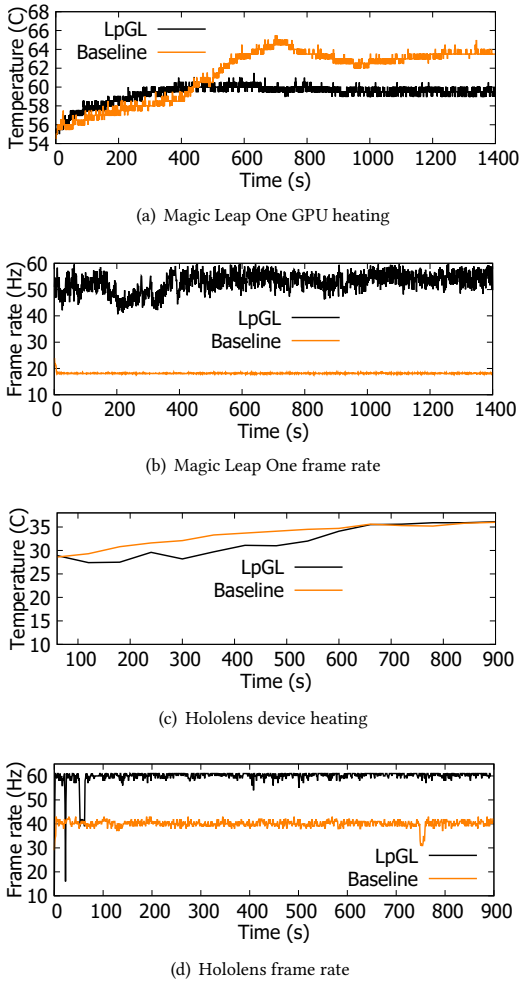
4.4 Latency Overhead

We now analyze the latency introduced by *LpGL*, in particular the latency it adds to the rendering pipeline. We found that the additional latency per-object for *LpGL* was 45.57μsec. As the number of objects increases, the overall latency of *LpGL* increases sub-linearly and was ~1.2ms for 32 objects and ~2.5ms for 64 objects. Potentially, we can distribute such operations over a number of threads to minimize latency impact induced by processing multiple objects.

In addition, the latency to perform *LpGL*'s run-time object simplification for dynamically loaded objects is 819 msec when simplifying a 69K triangle object to 50%. This long latency is fortunately a one time process when the object is first used. Later when the object is re-used, *LpGL* loads the pre-simplified object to the scene with minimal latency. Overall, the latency of *LpGL* is low and does not add any noticeable delay.

4.5 Heat Reduction

Finally, we also see device heating as another important factor to consider as mobile AR headsets are worn directly on the head, and high operation temperatures can negatively affect the usability. Therefore, we measure the temperature of the Magic Leap One



(a) Magic Leap One GPU heating

(b) Magic Leap One frame rate

(c) Hololens device heating

(d) Hololens frame rate

Figure 14: Temperature and achieved frame rate for the Magic Leap One and Microsoft Hololens

and the Microsoft Hololens with and without applying *LpGL* using a dynamic application that saturates the GPU's performance. Specifically, we record how the GPU/device temperature changes over time, along with their achieved frame rates and present the results in Figure 14 for Magic Leap One and Hololens. Note that the Magic Leap One's processing units are external to the headset. The users can carry an external processing unit wired to the headset device. On the other hand, on the Hololens, all processing units are integrated to the forehead component of the headset itself. Measuring heat on the Magic Leap One was done using its power and thermal profiler and we present the GPU temperature, which dominates the processing unit's temperature. As the Hololens does not provide such analysis software, we used an infrared thermal camera to capture its forehead processing unit temperature on a per-minute basis.

From Figures 14(a) and 14(b) we observe that, on the Magic Leap One, the GPU temperature of the baseline saturates at a higher

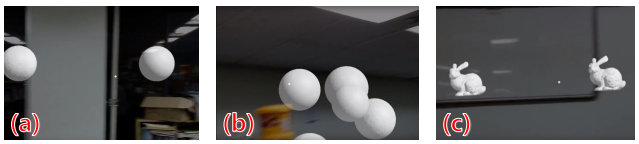


Figure 15: Screenshot from our three applications

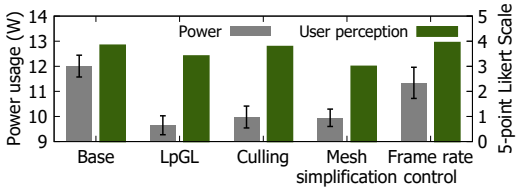


Figure 16: Mean power usage and user satisfaction scores in 5-point *Likert scale* for static scene app.

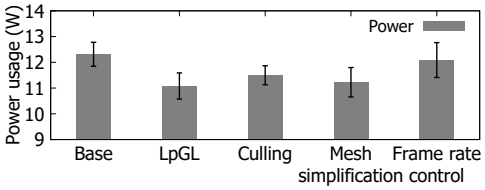


Figure 17: Mean power usage for dynamic/interactive scene application.

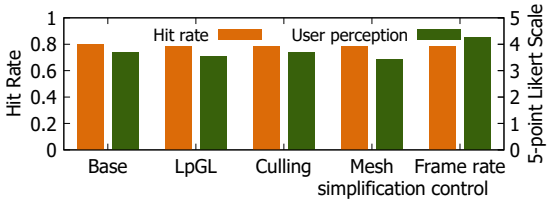


Figure 18: Hit rate and user perception in 5-point Likert scale for dynamic/interactive scene app.

level compared to *LpGL* and *LpGL*'s achieved frame rate is consistently higher. We see similar patterns for the Hololens (Figures 14(c) and 14(d)).

5 USER STUDIES

We now expand our evaluations to a set of IRB-approved user studies. Here, we analyze the power savings and usability of *LpGL* (and each of its features separately) under various scenarios. For this purpose, we use three types of applications: a static scene app, a dynamic/interactive scene app, and one where users were given tasks that relate to the fidelity of the objects displayed in the scene. Each experiment runs applications using five different configurations in randomized order for each participant: (1) with *LpGL*, (2) with *LpGL* culling only, (3) with *LpGL* frame rate control only, (4) with *LpGL* mesh simplification only, and (5) without *LpGL* (the native graphics rendering process). Our user study involved 25 participants (avg.age: 24.6, 10 female) and the overall time to complete an experiment (per-user) was ~30 mins with a $10 reward. At the end of each testing phase, users provided their subjective scores regarding the scene quality they experienced on a 5-point *Likert scale* (very poor:1, very good:5).

5.1 Static Scene App: Floating Spheres

The purpose of the static scene application (the first app presented to all participants), was to introduce and familiarize study participants to the mobile AR environment. No specific tasks were given while the users slowly gazed through the 16 spheres (69K triangles) that float (no dynamics) around the user (equally distributed over 360°). Each of the five configurations was used for 30 seconds with short breaks to answer user perception questions after each configuration.

Figure 16 presents the mean power usage and user perception results for each test case. It shows that *LpGL* can reduce power consumption by ~22% (~2.6 W) compared to the baseline graphics rendering process, and the mesh simplification and culling subcomponents contribute heavily to the power savings. This is because, for the static scene app, the spheres that are not in the participants' core FOV can be culled or simplified, which leads to a significant reduction in GPU and memory usage. From the system lifetime's perspective, the power reduction results for the baseline translates to 3.0 hours of operation time, and *LpGL* adds 0.9 additional hours of operation. Finally, we can see that the user perceived quality level reports for *LpGL* scored 3.54 compared to 3.80 for the baseline, but, this difference was not statistically significant (2-tailed t-test, $p<0.05$).

5.2 Dynamic Scene App: Sphere Shooting

The second application is a simple shooting game in which moving spheres fly into the scene, and participants are asked to use the controller and their gaze to "shoot" and eliminate the spheres. We design the application so that the spheres to appear in various patterns, in high-speeds, low-speeds, from the right and left, etc. Compared to static scenes, this application introduces both dynamics and interactive complexity. Again, we test with the five configurations mentioned above and measure the power usage, user perception level and the hit/miss rate of the sphere shooting performance to quantify task accomplishment levels.

Figure 17 plots the mean power consumption of this experiment with standard deviations. Results show that the frame rate component shows only minimal power reduction of 2%, noticeably less than 7% for the static scene app results in Figure 16. This is a result of having dynamic scenes, as *LpGL* tries to adapt to such scenes with high frame rates to preserve application quality. As in the static scene application case, the culling and frame rate control plays an important role; as the users focus on the spheres that they shoot, other spheres can be simplified or culled. Overall, *LpGL* saves ~11% power compared to the baseline consumption on average.

In Figure 18 we present the mean hit/miss rates of how accurate the users performed sphere shooting with the user perception levels. Despite changing the configurations, there is no noticeable change in the task accomplishment performance (i.e., hit rate) and the user perception is kept high across the different test cases.

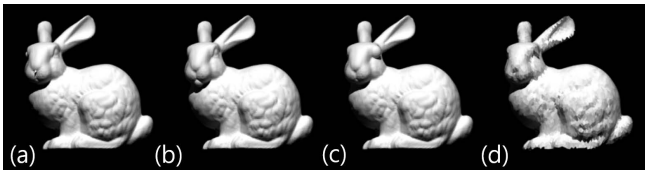


Figure 19: Abnormal bunnies designed for our user studies (a)-(c). Example of the reduced triangle bunny used for the naïve power reduction tests (d).

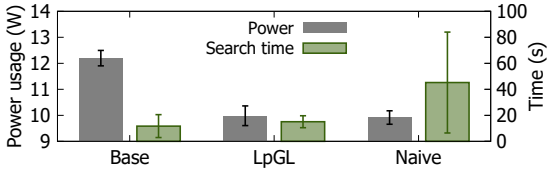


Figure 20: Mean power consumption and time consumed to accomplish abnormal bunny search task.

5.3 Fidelity-centric App: Bunny Search

The third application used in our user study was designed to conceptualize a group of real world AR applications in which the fidelity of the displayed object quality impacts task accomplishment performance. Specifically, as Figure 19 shows, we made small, but noticeable, changes to the original Stanford Bunny by adding small spheres to the eyes or mouth. In the scene, we uniformly distribute 16 bunnies (mixture of normal and abnormal) in the 360° space. Study participants would need to look close to the details of each bunny to identify the abnormalities. We then ask study participants to locate *two* abnormal bunnies with the same abnormality. These bunnies are located at least 120° away from each other to avoid cases where they are adjacent. We ask for two that are apart (instead of one) to avoid the case where an abnormal bunny is located closely at the view point at the beginning of the experiment; thus, easy to find with minimal effort. With such scenario, we run three experiments which include (1) *LpGL*, (2) baseline rendering, and (3) naïve power reduction. In the naïve power reduction case, we use the baseline rendering, but display bunnies with fewer triangles so that the power usage of the Magic Leap One matches *LpGL* (c.f., Figure 19). At the beginning of each test, we let the participant know what type of abnormality they are to find (among the three types), and record the power usage and search times until the task successfully completes. Note that prior to performing the three test cases, we allowed the participants to perform one practice run with the baseline rendering mode enabled, so that they well-understood the target task.

We present the mean power consumption and the task accomplishment times in Figure 20. When using *LpGL*, we save approximately 20% in energy compared to the baseline. At the same time, the time taken to accomplish the task was not noticeably affected (11 seconds for baseline 13 seconds for *LpGL*). On the other hand, when comparing *LpGL* against the naïve power reduction case, the power consumption of the two are similar, but the task completion time differ by more than three-fold. Given that the naïve power reduction case simplifies all displayed objects uniformly (see Figure 19(d)), identifying the abnormal bunny becomes bigger challenge than before, whereas *LpGL* reduces power usage while preserving the user perceived scene quality by exploiting the head and gaze orientation data from the mobile headset.

6 DISCUSSIONS AND FUTURE WORK

Applicability to other mobile AR headsets: *LpGL* is designed to be a device and platform-independent solution for reducing power usage on mobile AR headsets. Given that its design is centered around the standard graphics pipeline and native graphics library APIs (e.g., OpenGL), the implementation is easy to port for different platforms. In this work, while we mostly focus on the performance statistics of the Magic Leap One, as the results in Section 4.5 shows, *LpGL* is already implemented for use on the Microsoft Hololens as well. We note that we observed similar power usage reduction patterns on the Hololens by saving ~ 25% of power for static scenes and ~ 12% for dynamic scene applications. As part of our future work, we plan to expand *LpGL* support for different mobile AR platforms as well.

Power management of headset and other components: While our preliminary studies show that the headset itself is also a major consumer of power, this work focuses on the graphics pipeline-related aspects of mobile untethered AR platforms. Techniques such as those that duty-cycle the headset sensors and adaptively control the brightness/resolution of objects, combined with our efforts in optimizing the graphics pipeline, can have significant impact in increasing the lifetime of mobile AR headsets. However, the fact that the Magic Leap One, along with many other commercially available mobile AR headsets, expose only limited access to the headset configurations limits such research at this point.

In addition, the power usage of components such as WiFi need to be collectively managed. As mentioned in Section 2, findings from many previous work [1, 38, 39] can be applied to reduce WiFi power usage in mobile AR applications. Overall, we see this as an interesting direction of research towards designing a comprehensive framework for mobile AR headset power management.

Real-world application validation and user study limitations: Given that mobile AR headset development is still in its early stages, real-world applications for exploiting these platforms' full capabilities are still premature. At the same time, considering the impact that these new wearable computing devices can offer, the domains in which they can potentially contribute to can become very diverse. For this reason, while we wanted to test *LpGL* on a real-world, widely used application, it was difficult to define a single set of applications that were representative for mobile AR headsets. For this reason, our user studies focus on three types of applications, each with different application and scene characteristics. We do so to make sure that our evaluations cover diverse cases in how mobile AR headsets can contribute in novel applications. Nevertheless, the performance of *LpGL* and untethered mobile AR headsets in general will heavily depend on application complexity and its requirements, and we hope to apply *LpGL* on real-world applications as part of our future work.

Performance on image-heavy applications: There are many mobile AR applications that are mostly based on augmenting simple images or text (e.g., Apple Measure [42], Google Translate [43]), and potentially mobile AR headsets could be used for such applications. In such cases, usage of *LpGL* will not be limited, but the performance enhancements may be, given that these applications exploit low-levels of object geometry complexity.

We also point out that since *LpGL* is a geometry based approach, if contents within an object change (without changes in the boundaries), *LpGL* will not be able to capture such dynamics. Nevertheless, *LpGL* is conceptually orthogonal to approaches that exploit image-based (or frame-content-based) approaches to ease computation complexity (e.g., foveated rendering [18, 36], video streaming overhead reduction [11, 49, 50]), and these schemes can be exploited together with *LpGL* to conserve energy on both the image-level and geometry level. While we leave the system-level impact of such combination as future work, we see this as an interesting direction to further explore.

Animation support: 3D objects in an application may have different types of animation effects that complicate the geometry of an object over time. In this work, we validate the performance of *LpGL* for "rigid body annimation" effects (e.g., sphere/bunny moving its coordinates without any internal changes). Furthermore, we expect *LpGL* to be effective for animation effects with "affine transformation" as well, which involves translation, rotation and scaling of objects, given that the RDCC engine is designed to consider objects' transformation information. Unfortunately, we have not validated the usability of other types of animation techniques such as shape-deformable or image-based animations (e.g., skinning, particle system animation, texture animation [32, 35], fluid dynamics animation, human-body animation [35]). We see the combination of *LpGL* with retargetting [15] and morphing [28] to be effective to resolve such issues and plan to pursue additional research in this space as part of our future work.

7 RELATED WORK

A number of prior work exploit human perception information to reduce computation or energy consumption while minimizing sacrifice in user experience on mobile headsets. Guenter et al. [18] presented a way to improve performance by using eye tracking information to render backgrounds with low resolution and focused-content with high resolution. Patney et al. at NVIDIA has also announced an improved foveated rendering technique for VR displays which optimizes performance by reducing image quality in the viewer's peripheral vision based on gaze tracking [36]. Hwang et al. proposed a PSNR-based method to reduce energy consumption using frame rate scaling by comparing the contents of two consecutive frames in mobile games [23]. Tan et al. proposed FocusVR [47], which performs screen dimming and vignetting to reduce VR headset power consumption. While sharing similar goals with *LpGL*, these works focus on the VR environment, where background rendering plays a significant role in power usage reduction. Our work is tailored towards mobile AR headsets; thus, addresses different challenges such as those discussed in Section 2.1.

LpGL resides between the application and the graphics library, providing a transparent compatible layer that can intercept OpenGL commands. This is similar to the approach taken by Miao et al. in [34]. However, their focus was to adapt the graphics stack to the circular displays of wearable smart watches and reduce the memory and display interface traffic wasted due to non-rectangular displays, while our approach focuses on energy efficiency.

In an attempt to reduce energy usage of the display on a mobile device without deteriorating user perception, LPD [19] reduces the memory and display interface traffic by utilizing the display update information to suppressing the update of unchanged parts. He et al. [20] makes an observation that a reduced display resolution may still achieve the same user experience when the user-screen distance is large. Based on this idea, authors adopt an ultrasonic-based approach to accurately detect the user-screen distance and make dynamic scaling decisions for maximum user experience and power saving. Anand et al. [4] utilize the pixel brightness to save significant amounts of power while preserving image qualities, and FingerShadow [8] performs local dimming for the screen areas covered by users' fingers to save power without compromising their visual experiences. Chameleon [10] reduces power consumption for web browsing task on an mobile OLED-based device in color-adaptive way to preserve real-time task capability. However, these work are in the context of mobile devices and do not take into account the characteristics of untethered AR headsets.

8 CONCLUSION

We presented *LpGL*, an OpenGL API compatible Low-power Graphics Library for energy efficient untethered mobile AR headset application development. We first presented extensive measurements detailing the power consumption characteristics of the Magic Leap One. From there, we built *LpGL* to use gaze/head orientation and geometry data to obtain user perception information, and exploit this to adaptively apply frame rate scaling, mesh simplification, and culling techniques to enhance the battery lifetime of untethered AR headsets while minimizing losses in user perceived scene quality. Moreover, these features are fully application-layer transparent. Through comprehensive controlled in-lab experiments and an IRB-approved 25 participant user study, we showed that *LpGL* can reduce power consumption by up to ~22%, with minimal latency and user experience impact.

ACKNOWLEDGEMENTS

We would like to thank the anonymous reviewers and our shepherd Dr. Robert LiKamWa for providing insightful comments on improving the quality of the paper. We also thank Jaehee Choi for providing the illustrations used in this work. This work was supported by the the Basic Science Research Program through the National Research Foundation of Korea funded by the Ministry of Science and ICT (2018R1C1B6003869).

REFERENCES

[1] Omid Abari. 2017. Enabling High-Quality Untethered Virtual Reality. In *Proceedings of the 1st ACM Workshop on Millimeter-Wave Networks and Sensing Systems 2017 (mmNets '17)*. ACM, New York, NY, USA, 49–49. https://doi.org/10.1145/3130242.3131494

[2] Tomas Akenine-Moller, Eric Haines, and Naty Hoffman. 2008. *Real-Time Rendering* (3rd ed.). A. K. Peters, Ltd., Natick, MA, USA.

[3] Farhan Azmat Ali, Pieter Simoens, Tim Verbelen, Piet Demeester, and Bart Dhoedt. 2016. Mobile device power models for energy efficient dynamic offloading at

runtime. *Journal of Systems and Software* 113 (2016), 173 – 187.
[4] Bhojan Anand, Karthik Thirugnanam, Jeena Sebastian, Pravein G. Kannan, Akhihebbal L. Ananda, Mun Choon Chan, and Rajesh Krishna Balan. 2011. Adaptive Display Power Management for Mobile Games. In *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services (MobiSys'11)*. 57–70.
[5] Edward Angel and Dave Shreiner. 2011. *Interactive Computer Graphics: A Top-Down Approach with Shader-Based OpenGL* (6th ed.). Addison-Wesley Publishing Company, USA.
[6] Vivek D. Bhise. 2011. *Ergonomics in the Automotive Design Process*. CRC Press.
[7] Inter IKEA Systems B.V. 2018. IKEA Place. Available at https://www.ikea.com/gb/en/customer-service/ikea-apps.
[8] Xiang Chen, Kent W. Nixon, Hucheng Zhou, Yunxin Liu, and Yiran Chen. 2014. FingerShadow: An OLED Power Optimization Based on Smartphone Touch Interactions. In *USENIX 6th Workshop on Power-Aware Computing and Systems (HotPower'14)*.
[9] X. Corbillon, G. Simon, A. Devlic, and J. Chakareski. 2017. Viewport-adaptive navigable 360-degree video delivery. In *2017 IEEE International Conference on Communications (ICC)*. 1–7. https://doi.org/10.1109/ICC.2017.7996611
[10] Mian Dong and Lin Zhong. 2011. Chameleon: A Color-adaptive Web Browser for Mobile OLED Displays. In *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services (MobiSys '11)*. ACM, New York, NY, USA, 85–98. https://doi.org/10.1145/1999995.2000004
[11] Ching-Ling Fan, Jean Lee, Wen-Chih Lo, Chun-Ying Huang, Kuan-Ta Chen, and Cheng-Hsin Hsu. 2017. Fixation Prediction for 360&Deg; Video Streaming in Head-Mounted Virtual Reality. In *Proceedings of the 27th Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV'17)*. ACM, New York, NY, USA, 67–72. https://doi.org/10.1145/3083165.3083180
[12] Michael Garland and Paul S. Heckbert. 1998. Simplifying Surfaces with Color and Texture Using Quadric Error Metrics. In *Proceedings of the Conference on Visualization '98 (VIS '98)*. IEEE Computer Society Press, Los Alamitos, CA, USA, 263–269. http://dl.acm.org/citation.cfm?id=288216.288280
[13] Nirit Gavish, Teresa GutiÃlrrez, Sabine Webel, Jorge RodrÃŋguez, Matteo Peveri, Uli Bockholt, and Franco Tecchia. 2015. Evaluating virtual reality and augmented reality training for industrial maintenance and assembly tasks. *Interactive Learning Environments* 23, 6 (2015), 778–798.
[14] Google Glass. 2018. Available at https://x.company/glass/. Last accessed 2018-04-02.
[15] Michael Gleicher. 1998. Retargetting Motion to New Characters. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '98)*. ACM, New York, NY, USA, 33–42. https://doi.org/10.1145/280814.280820
[16] Theodore P. Grosvenor. 2007. *Primary Care Optometry*. Elsevier Health Sciences.
[17] Khronos Group. 2018. OpenGL ES 3.0 - OpenGL for Embedded Systems version 3.0. Available at https://www.khronos.org/opengles/. Last accessed 2018-04-02.
[18] Brian Guenter, Mark Finch, Steven Drucker, Desney Tan, and John Snyder. 2012. Foveated 3D Graphics. *ACM Trans. Graph.* 31, 6, Article 164 (Nov. 2012), 164:1–164:10 pages.
[19] MyungJoo Ham, Inki Dae, and Chanwoo Choi. 2015. LPD: Low Power Display Mechanism for Mobile and Wearable Devices. In *USENIX Annual Technical Conference (USENIX ATC 15)*. 587–598.
[20] Songtao He, Yunxin Liu, and Hucheng Zhou. 2015. Optimizing Smartphone Power Consumption Through Dynamic Resolution Scaling. In *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking (MobiCom'15)*. 27–39.
[21] Microsoft HoloLens. 2018. Available at https://www.microsoft.com/en-us/hololens. Last accessed 2018-04-02.
[22] John F. Hughes, Andries van Dam, Morgan McGuire, David F. Sklar, James D. Foley, Steven Feiner, and Kurt Akeley. 2013. *Computer Graphics: Principles and Practice* (3 ed.). Addison-Wesley, Upper Saddle River, NJ.
[23] Chanyou Hwang, Saumay Pushp, Changyoung Koh, Jungpil Yoon, Yunxin Liu, Seungpyo Choi, and Junehwa Song. 2017. RAVEN: Perception-aware Optimization of Power Consumption for Mobile Games. In *Proceedings of the 23rd Annual International Conference on Mobile Computing and Networking (MobiCom'17)*. 422–434.
[24] Augmedix Inc. 2018. Augmedix. Available at https://www.augmedix.com.
[25] Magic Leap Inc. 2018. Available at https://www.magicleap.com/magic-leap-one. Last accessed 2018-04-02.
[26] Niantic Inc. 2018. Pokemon Go. Available at https://www.pokemongo.com.
[27] Tan Kiat Wee, Eduardo Cuervo, and Rajesh Krishna Balan. 2016. Demo: FocusVR: Effective & Usable VR Display Power Management. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services Companion (MobiSys '16 Companion)*. ACM, New York, NY, USA, 122–122. https://doi.org/10.1145/2938559.2938564
[28] Aaron W. F. Lee, David Dobkin, Wim Sweldens, and Peter Schröder. 1999. Multiresolution Mesh Morphing. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '99)*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 343–350. https://doi.org/10.1145/311535.311586
[29] Robert LiKamWa, Zhen Wang, Aaron Carroll, Felix Xiaozhu Lin, and Lin Zhong. 2014. Draining Our Glass: An Energy and Heat Characterization of Google Glass. In *Proceedings of 5th Asia-Pacific Workshop on Systems (APSys '14)*.
[30] K. W. Lim, J. Ha, P. Bae, J. Ko, and Y. B. Ko. 2018. Adaptive Frame Skipping With Screen Dynamics for Mobile Screen Sharing Applications. *IEEE Systems Journal* PP, 99 (2018), 1–12. https://doi.org/10.1109/JSYST.2016.2589238
[31] F. Liu, P. Shu, and J. C. S. Lui. 2015. AppATP: An Energy Conserving Adaptive Mobile-Cloud Transmission Protocol. *IEEE Trans. Comput.* 64, 11 (Nov 2015), 3051–3063.
[32] Frank Luna. 2012. *Introduction to 3D Game Programming with DirectX 11*. Mercury Learning & Information, USA.
[33] Autodesk Maya. 2018. Available at https://help.autodesk.com/cloudhelp/2017/CHS/Maya-Tech-Docs/CommandsPython/polyReduce.html. Last accessed 2018-12-14.
[34] Hongyu Miao and Felix Xiaozhu Lin. 2016. Tell Your Graphics Stack That the Display Is Circular. In *Proceedings of the 17th International Workshop on Mobile Computing Systems and Applications (HotMobile'16)*. 57–62.
[35] Rick Parent. 2012. *Computer Animation: Algorithms and Techniques* (3 ed.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.
[36] Anjul Patney, Marco Salvi, Joohwan Kim, Anton Kaplanyan, Chris Wyman, Nir Benty, David Luebke, and Aaron Lefohn. 2016. Towards Foveated Rendering for Gaze-tracked Virtual Reality. *ACM Trans. Graph.* 35, 6, Article 179 (Nov. 2016), 179:1–179:12 pages.
[37] Microsoft Universal Windows Platform. 2018. What's a Universal Windows Platform (UWP) app? https://docs.microsoft.com/en-us/windows/uwp/get-started/universal-application-platform-guide.
[38] Feng Qian, Bo Han, Qingyang Xiao, and Vijay Gopalakrishnan. 2018. Flare: Practical Viewport-Adaptive 360-Degree Video Streaming for Mobile Devices. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking (MobiCom '18)*. ACM, New York, NY, USA, 99–114. https://doi.org/10.1145/3241539.3241565
[39] Feng Qian, Bo Han, Qingyang Xiao, and Vijay Gopalakrishnan. 2018. Tile-Based Viewport-Adaptive Panoramic Video Streaming on Smartphones. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking (MobiCom '18)*. ACM, New York, NY, USA, 817–819. https://doi.org/10.1145/3241539.3267715
[40] Moo-Ryong Ra, Jeongyeup Paek, Abhishek B. Sharma, Ramesh Govindan, Martin H. Krieger, and Michael J. Neely. 2010. Energy-delay Tradeoffs in Smartphone Applications. In *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services (MobiSys'10)*. 255–270.
[41] Hans Strasburger, Ingo Rentschler, and Martin JÃijttner. 2011. Peripheral vision and pattern recognition: A review. *Journal of Vision* 11, 5 (2011), 13. https://doi.org/10.1167/11.5.13 arXiv:/data/journals/jov/933487/jov-11-5-13.pdf
[42] Use the Measure app on your iPhone or iPad. 2019. Available at https://support.apple.com/en-us/HT208924. Last accessed 2019-03-29.
[43] Google Translate. 2019. Available at https://translate.google.com. Last accessed 2019-03-29.
[44] Greg Turk and Marc Levoy. 1994. "Stanford Bunny". Available at Stanford University Computer Graphics Laboratory http://graphics.stanford.edu/data/3Dscanrep/.
[45] X. Wang, S. K. Ong, and A. Y. C. Nee. 2016. A comprehensive survey of augmented reality assembly research. *Advances in Manufacturing* 4, 1 (01 Mar 2016), 1–22.
[46] Zhou Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. 2004. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing* 13, 4 (April 2004), 600–612.
[47] Tan Kiat Wee, Eduardo Cuervo, and Rajesh Balan. 2018. FocusVR: Effective 8 Usable VR Display Power Management. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 2, 3, Article 142 (Sept. 2018), 25 pages. https://doi.org/10.1145/3264952
[48] Wikipedia. 2018. Available at https://en.wikipedia.org/wiki/Peak_signal-to-noise_ratio. Last accessed 2018-12-14.
[49] Shaowei Xie, Qiu Shen, Yiling Xu, Qiaojian Qian, Shaowei Wang, Zhan Ma, and Wenjun Zhang. 2018. Viewport Adaptation-Based Immersive Video Streaming: Perceptual Modeling and Applications. arXiv:arXiv:1802.06057
[50] Alireza Zare, Alireza Aminlou, Miska M. Hannuksela, and Moncef Gabbouj. 2016. HEVC-compliant Tile-based Streaming of Panoramic Video for Virtual Reality Applications. In *Proceedings of the 24th ACM International Conference on Multimedia (MM '16)*. ACM, New York, NY, USA, 601–605. https://doi.org/10.1145/2964284.2967292