Gamelets - Multiplayer Mobile Games with Distributed Micro-Clouds

Bhojan Anand, Aw Jia Hao Edwin
School of Computing, National University of Singapore
email: banand@comp.nus.edu.sg, faets@live.com.sg

***Abstract*—In recent years, cloud computing services have been increasing in greater pace. High penetration rate of mobile devices and resource limited devices escalate the demand for cloud services further [1], [2]. Even though the cloud industry continues to grow exponentially, the cloud gaming service has been left behind due to the limitations in today's technology. There are three well known reasons for the slower growth - latency, server scalability (esp. bandwidth) and lack of game data at client side to use latency hiding and synchronisation techniques such as Dead-reckoning. In this paper, we propose a novel distributed micro-cloud infrastructure with a next generation device called Gamelet to mitigate the limitations in traditional cloud system for multiplayer cloud gaming on resource limited mobile devices. The paper also investigates the opportunities, issues and possible solutions for Gamelet infrastructure for mobile games with a demonstrable prototype.**

I. Introduction

Cloud computing conventionally follows three fundamental models, Infrastructure as a Service(IaaS) , Platform as a Service (PaaS) and Software as a Service (SaaS). At present, although it is still a relatively young, cloud computing has proven to be a very welcomed system, with revenue from Software-as-a-Service (SaaS) alone reached an astonishing $14.5 billion in 2012 [3] and forecasted to grow five times faster than traditional software packages [4].

Lately games have also started to make an appearance into the cloud scene with Games on Demand (GoD) which has tapped onto the SaaS type of cloud technology. A cloud gaming system must collect a player's actions, transmit them to the cloud server, process the action, render the results, encode the resulting changes to the game world, compress, and stream the video (game scenes) back to the player. To ensure interactivity, all of these serial operations must happen within milliseconds. Intuitively, this amount of time, which is defined as interaction delay, must be kept as short as possible in order to provide a rich Quality of Experience (QoE) to cloud game players. This latency should be less than 100ms for FPS games, 500ms for RPG games and 1000ms for RTS games [5]. This makes cloud gaming one of the most challenging multimedia application. There are two distinct methods used by cloud gaming services in general. In the first method, the game is played on a virtual machine and the rendered results are compressed and streamed. In the second method, streaming is built-in into the game itself. However, these conventional cloud gaming approaches suffer from three fundamental issues - latency [5], server scalability (in terms of bandwidth, computation, memory) and lack of game data and game knowledge at the client side to use latency hiding and synchronisation techniques such as Dead-reckoning [6] and Local Perception Filters [7] for smooth gameplay. *While traditional client/server games are extensively relying on various latency hiding techniques to provide good multiplayer gaming experience, current Cloud games system makes these techniques hard to use, despite introducing additional latency for cloud processing and encoding*. Although the rendering and encoding latency will likely fall with faster hardware encoders, a significant portion of network latency is unavoidable as it is bounded by the speed of light in fibre [8].

In this work, we aim to improve performance of Cloud games for resource limited mobile devices and alleviate challenges faced by game developers, by proposing the development of a next generation infrastructure called Gamelet system. *Gamelet* system is basically a distributed micro-cloud system in which the computational intensive tasks are offloaded to a Gamelet node that is few hops (most of the time one or two hops) away from the mobile client. With Gamelets the bandwidth and processing requirement of a game server (which may be running in traditional Cloud system) will be same as traditional game servers while the benefits of cloud system can be availed.

II. Related Works and Motivations

There are two distinct methods used by cloud gaming services in general. The first method, which is more conventional SaaS, consists of remote gaming on virtual machines and streaming the viewport of the virtual machine to the users. Users are granted their own game cloud, which stores all the games they have access to. This allows users who access to a large set of games on their game clouds and users can play from any location with a good connection with the providers central servers. This approach, which has been implemented by cloud gaming giants like Gaikai, has been growing in popularity over the past years in locations, which it is supported. Gaikai currently owns a Guinness world record of fifty million gamers per month [1].

The main disadvantage to this approach is the need to be in relatively close proximity from the provider's central servers [9]. Consumers, who are further away, will either be affected by much higher latency making most games intolerable, or will be prohibited from connecting to the server. Thus, global use of this type of systems will require the introduction of servers in a large number of regions, making it cost inefficient [10]. With cellular networks network latency goes up to 200ms [5] even with very close proximity and these mobile clients are completely eliminated from playing Cloud games.

[1] http://www.gaikai.com/

The second method is to have the streaming technology built into games. For convenience sake, we will call the games developed this way 'pure cloud games'. Pure cloud games are usually played on a browser. Users connect to game servers on their browsers and are fed a stream (video/image) of what is happening in the game. This method is supported by a large array of devices and has been extremely successful on social media platforms such as Facebook [11]. The current problem with this approach is the genre of games that can be played is greatly limited. At present, only games, which are not affected by high latency and delays such as turn-based games, can effectively market on this approach. This heavily limits the types of games, which can be developed on this paradigm.

Apart from latency, in both methods server scalability is affected due to high bandwidth (also, processor and memory) requirement to stream image/video instead of small game update packets of traditional client/server games. In addition, with cloud based games the client is just a video/image rendering and UI device and it cannot run additional algorithms to hide latency as the game data is not available at the client side. We introduce Gamelet system to mitigate these drawbacks.

Distributed Rendering: There are several solutions to distributed rendering, with the most prominent one being, dividing a viewport into multiple sections and rendering each section in parallel using a different core, processor, graphic card, or device. Figure 1 illustrates how the rendering is divided into sections by Digipede's [12] parallel rendering solution. On the left, the entire viewport is rendered sequentially using only one device. On the right, the viewport is broken down into 25 sections and rendered in parallel on multiple devices, increasing the speed of rendering exponentially as more devices are dedicated to it.

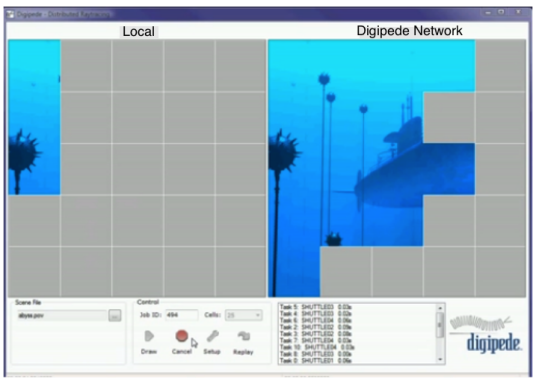

Fig. 1. Distributed Rendering via Parallel Processing

III. Gamelet Vision & Challenges

Gamelet is a minimal-set hardware required to run, render and stream 3D games placed in the same local network or few hops (most of the time one or two hops) away from the mobile client. Adjacent Gamelets running the same game can communicate with each other for information sharing and distributed rendering for increased efficiency. Gamelets run client portion of the game with conventional latency hiding and synchronising techniques such as Dead-reckoning [6] and Local Perception Filtering [7] to provide good QoE.

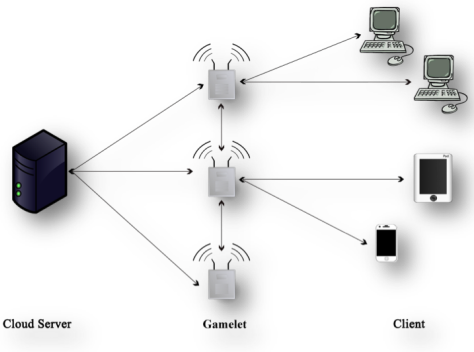

Fig. 2. Gamelet Communication Structure

The Gamelet is designed to take over the rendering task from game cloud servers.The Gamelet can process the players' actions and give players immediate feedback, which helps to mask the latency between the Gamelet and the central game servers where the players' actions are processed. The image or video are streamed only in the local wireless network which means each user will be able to receive a larger dedicated bandwidth, without incurring larger Internet costs. In addition it reduces the transmission time of the image/video frame and improves network latency.

We envision Gamelet as a modified version of today's ever-common public WiFi access points with some additional hardware (Figure 2). However, Gamelet can be separate hardware connected to the local wireless network access points or power full peer game client rendering for itself and other players. For example, a Gamelet can be a common processing box in the home network which can stream game to multiple devices in the home. Proposed basic hardware includes: *Processing Unit, RAM , Graphics Card, Flash Drive (For lightweight operating system).*

1) Deployment: Gamelets can be deployed and managed by third party (not the game developers) service providers to provide subscription based value added service to their clients. For example, public/private WiFi service provider (shopping complex, airport lounge or coffee shop management). The mobile client playing the streamed game can be very thin with a capability to play the video stream provided by the Gamelet and transfer the user inputs to the Gamelet. In an extreme case the client player can be simply a display or projector device with some interface for human interaction (eg. touch sensors or hand movement sensors). For example, the game can be simply projected on the wall and the player can play by directly manipulating (with hand/finger sensor) the displayed contents. The Gamelet system can provide resource adaptive resolution. For example, a power full Gamelet or set of connected Gamelets can provide resolution beyond the mobile screen resolution for big displays adaptively depending on the amount of free resources they have.

2) Key Challenges of the Gamelet System: Overall, the Gamelet system looks very interesting and promising solution

to improve the efficiency of cloud games. However, the system has some key challenges. The challenges and our initial solutions are given below.

- *Zone Distribution.* 3D games of today require a lot of data to run. The average size of present day 3D games is approximately 5 Gbytes. For example, the recommended system configuration for Battlefield 3, a highly popular FPS game is 4 GB RAM and 20 GB storage space [5]. Hence, the gamelet may not be able to host more than a 1-2 games. Also, it takes up to 56 minutes to download a game, assuming 1.5 Mbps bandwidth. We address this by dividing the game world and game resources to zones. Each Gamelet download the zone in which its client is playing. In addition, adjacent Gamelets holding different zones share the rendered data to improve efficiency. For example, if two players are playing different zones in a game served by adjacent Gamelets and they swap zones, the rendered zone data will be exchanged between Gamelets instead of downloading and rendering again.
- *Distributed Rendering.* 3D computer graphics improves rapidly over time, how will the Gamelet prevent itself from becoming obsolete too quickly? Distributed rendering helps to slow down the process of obsoleting. By exploiting the interconnectivity between Gamelets, powerful adjacent Gamelets can provide rendering assistant to old less powerful ones.
- *Security.* The biggest challenge is security, loss of centralised control and cheating. This is not new, the problem is similar to traditional client/server games and there are significant amount of research to address most of the issues [13][14][15]. In fact, the problem can be minimised - the problems should be handled only up to Gamelet tire not up to the client device level.
- *Content Based Adaptive Streaming.* Unlike video streaming, games have strict realtime constraints and hence more effective and faster compression and encoding algorithms required to minimise bandwidth requirement per client while improving the end-user interaction latency and QoE. In addition to the use of state-of-the art image/video compression and encoding techniques, we have designed Content Based Adaptive Streaming (CBAS) which exploits the properties of the Game content to reduce the bandwidth usage further. For example, static game regions are streamed at a lower frame rate.

We have developed a basic prototype to test and evaluate. The proposed system opens up huge set of research challenges (listed in Section VI) for large scale implementation, some of which are also discussed below.

IV. Implementation

A. Zone distribution

Zone distribution divides the game world into a graph of nodes. Each zone consists of all the information required for a user to play the game in his specific zone. This includes the player's character models, skybox, terrain map and others (Figure 3). Zone size is determined based on the following factors: 1) Time to download the zone; 2) Time to load the zone. Zone division procedure is given in Algorithm 1 and Figure 4 visually depicts the procedure. We can resize the Zones dynamically based on network conditions by running the algorithm periodically. Zone request is processed as follows by each Gamelet: 1)When user's camera enters the boundary (Figure 5) of the new zone a 'request to download' is issued; 2) When the user's far clip plane enters a new zone a 'request to load' is issued.

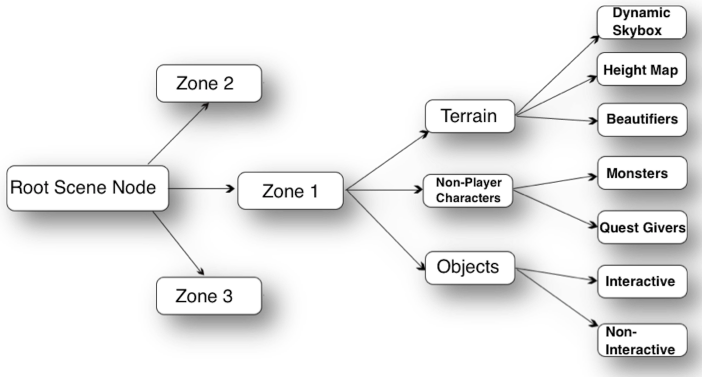

Fig. 3. Zones and what they contain

Algorithm 1 Game World Division

input: α = Estimated worst case RTT
input: β = Estimated worst case bandwidth per user
input: μ = Desired worst case total download time

maxZoneSize = $\beta(\mu + \alpha)$
Divide map into 4 zones over x,z axis about origin
if objects centre lie on the axis itself **then**
    distribute to either side of axis based on how many objects on either side
end if
Push zones into queue
while queue not empty **do**
    Current zone = Pop top of queue
    if Current zone size > maxZoneSize **then**
        Subdivide zone by 4 again
        Push all 4 zones back into queue
    end if
end while

Finding Boundaries: Boundaries are the places in the world which prompts the game to request additional game data for downloading. The game will prompt a new zone download when the player's far clip camera enters the boundary area. The algorithm (Algorithm 2) to find boundaries is quite simple and is called after the zones are divided. With this algorithm, we ensure that a zone is fully downloaded by the time the player camera can render it.

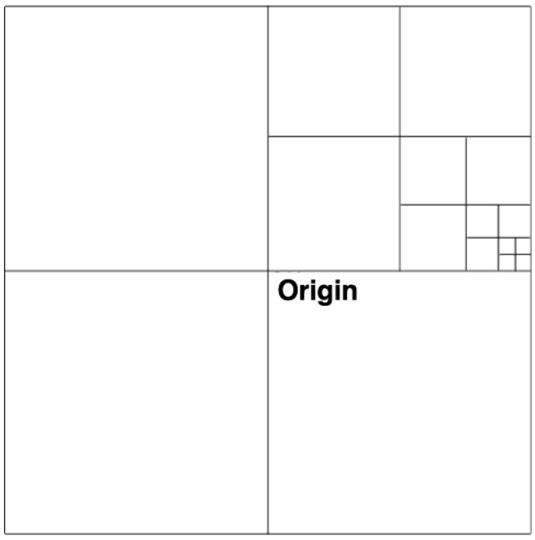

Fig. 4. Zone Division (Visual illustration)

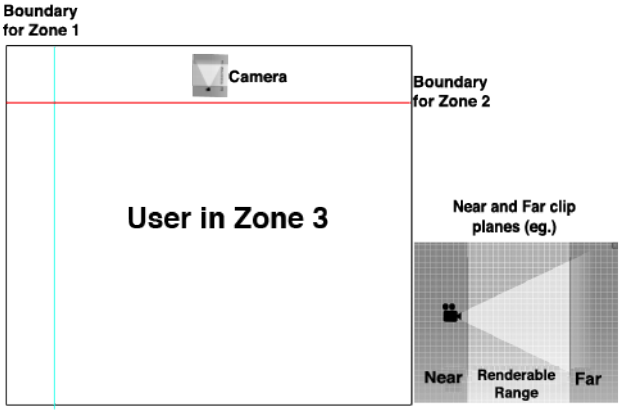

Fig. 5. Zone Division - Zone Boundary

B. Distributed Rendering

Most distributed rendering frameworks build on two primary modules: the Real-Time Scene Graph (RTSG) [16] and the Network-Integrated Multimedia Middleware (NMM) [17]. And at present all of the top used game engines (Unity [18], Unreal [19], RAGE [20] , cryENGINE [21]) do not have the NMM layer, and do not allow developer access to the RTSG layer so developers are not able to implement their own NMM layers. Games today are developed in a way that they can be rendered at a playable frame rate on the minimum hardware specifications of the game, so there is little motivation in investing in developing a distributed rendering system. Thus, this paradigm of distributed rendering is not usable if a developer chooses to use a game engine to develop his game.

Although the RTSG layer of the development architecture cannot be accessed by game developers developing on game engines, there still exists a unique workaround catered specifically for the Gamelet. Distributed rendering can still be done

Algorithm 2 Finding Boundaries

input: $\mu =$ Desired worst case total download time
input: $\phi =$ Player far clip plane distance
input: $\omega =$ Distance player covers in 1 second of movement in game

for all zones **do**
BoundarySize == $\phi + \omega / \mu$)
Propagate zone edges and boundary size to all neighbouring zones
end for

by reducing the workload on one Gamelet rendering engine and increasing the workload on another, without accessing the RTSG layer. We reduce the camera view area, use more cameras and then render each camera view in different Gamelet. By decreasing the view of a camera, we indirectly decrease the workload of the rendering engine (refer Table II). Compression time for rendered images decreases linearly. We call this method as *Multi-Camera Distributed Rendering* (MCDR).

C. Multi-Camera Distributed Rendering (MCDR)

There are two ways to do multi-camera distributed rendering. We will discuss them shortly here.

1) Rotating Camera: The concept of rotating the camera is very simple, by reducing its aspect ratio the camera naturally renders a smaller number of pixels so the individual work done on a gamelet is reduced. The missing pixels are then rendered by assisting gamelet with cameras positioned at the same location but rotated in the direction of the missing pixels (Figure 6). This method however, does not provide a perspectively correct image. This is due to the fact that the near and far clip planes on the cameras view frustums are not aligned between all rendering units. The result of rendering using this method is an inverted fish eye view of the world. While objects which are far away seem to be rendered correctly, once the objects are moved closer to the near clip plane, the inverted fish eye effect becomes instantly distinct (Figure 7). So this method can't be used.

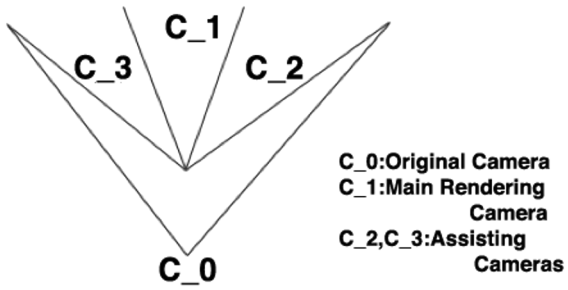

Fig. 6. Illustration of rotating cameras

2) Reshaping the view frustum: The most appropriate way to perform multi camera distributed rendering is by reshaping the view frustum. This can be done and synchronised across game worlds by setting up a camera with the same settings as

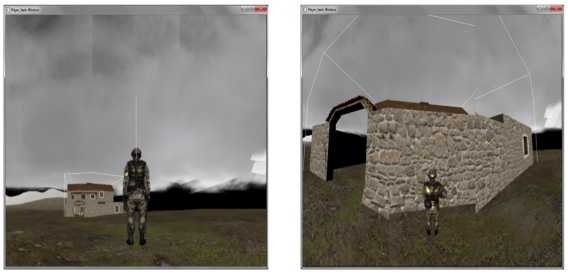

Fig. 7. 3D illustration of fish eye view on objects near and far

the original camera (not distributed), which means same aspect ratio, near and far clip planes (Figure 8). And by manipulating the location which the side clip planes intersect the near clip plane, you will be able to render the exact same image as the original camera at the same aspect ratio across multiple cameras.

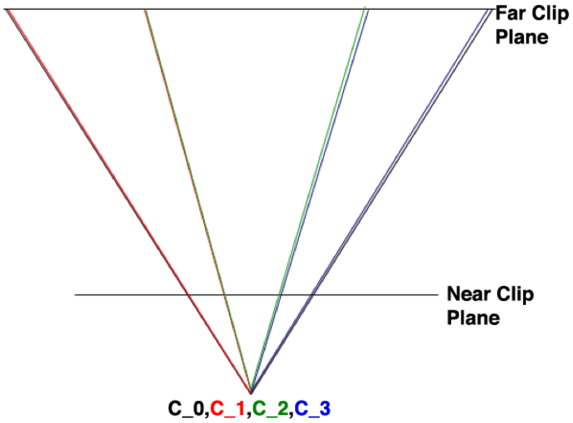

Fig. 8. Perspectively correct multi camera rendering

D. Content Based Adaptive Streaming

CBAS (Content Based Adaptive Streaming) uses the following techniques to reduce the bandwidth requirement per client. Based on our analysis, in most of the popular Games about 30% of the display is used for in-game HUD (Head-up-Displays). HUDs display the player's vital information such as health, ammo and equipment (Figure 9). As HUD area is relatively static, we stream this area at much lower rate. For example, 5-8 frames per second instead of usual 25-30 frames per second. Similarly, when the player is not doing any action and his view is static, we stream the player's background much lower frame rate. This background area, called as *Non-key region*, is shown in Figure 10. In addition, when *game state* is not important (for example, player is in idle state or slowly moving state and no other players around him to interact), we stream the game in lower rate 15-20 frames per second. Our previous works list several techniques for identifying the importance of game state [22][23][24][25]. The game state is computed at the game server. Study on trade-off in computing game state and its benefits in Gamelet system are deferred to future work.

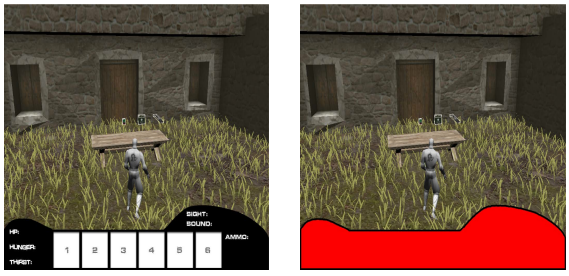

Fig. 9. In-game HuD area

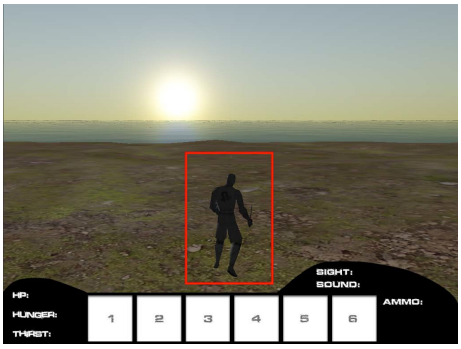

Fig. 10. Background region outside the red box (Non-key region)

E. Selecting Adjacent Gamelet

Adjacent Gamelet are selected for rendering assistance based on three key parameters, network latency, rendering latency and expected frame rate. At least 25 fps is required for fluidity of the game play. Hence, the maximum rendering latency is fixed to $\frac{1}{25}$s which is 40 ms. Which means the Gamelet should send data to adjacent helping Gamelet, get it rendered by the helper and receive back the rendered data within 40 ms. Each Gamelet continuously maintain a neighbour list of adjacent Gamelet which satisfy this latency constraint. Gamelet will get their initial list from the Game server which filters the gamelet based on IP-to-geo location mapping. Individual Gamelet use this initial list to build their adjacency list based on round-trip latency to other Gamelet in the list. Alternatively, to keep it simple, the adjacent Gamelet search can be restricted to same subnet.

V. Evaluation

We developed a test game on the architecture of the Gamelet system and focused our tests on zone distribution and distributed rendering. Due to lack of space, we present evaluations with respect to processing efficiency, bandwidth requirements and user perception of the Gamelet system.

1) Bandwidth: Bandwidth requirement for streaming images from Gamelet to mobile client (resolution: 800x480 WVGA) is around 1 to 3 Mbps after state-of-the-art compression with CBAS. Efficiency of CBAS depends on the size of the HuD and player's state (eg. idle and static view). Table I shows performance of a CBAS algorithm on an uncompressed game screen shot. Though there is an average of 3ms to 5ms processing overhead introduced by CBAS, it reduces bandwidth requirement effectively. (Note: If the game

is streamed at 25 frames per second and the HuD area is streamed at 5 frames per second, effectively HuD transmission is removed from 20 game frames.)
With 1 to 3 Mbps per client, up to five clients can be supported per access point over 11 Mbps 802.11b networks. However, the bandwidth to send game update (20 updates per second each $\tilde{6}0$ bytes) from game server to a Gamelet is less than 9.6 Kbps. If we use conventional cloud game system this server bandwidth will escalate up to 3 Mbps per client which severely affects the scalability of the server. However, when there more than 2 clients connected to a Gamelet and if there is no helper Gamelets nearby, the frame rate drops significantly. This can be mitigated with increasing the configuration of the Gamelet, installing more Gamelets for peer assistance or hosting a Gamelet process in one or more powerful game clients.

TABLE I
CBAS COMPRESSION AND OVERHEADS ON AN UNCOMPRESSED GAME SCREEN SHOT (AVERAGE VALUES)

Process	**Overhead**	**Data size reduction**
In-Game HUD Removal	3ms	17%
Non-Key Region Removal	5ms	78%

2) Processing Efficiency: We used windows task manager to measure CPU usage. GPU performance was recorded using MSI Afterburner version 2.3.1. In each run for measurement, the user took the same route through the game and looks at the same objects. He then comes to a stop and allows the hardware monitors to level off to determine the average CPU & GPU usage. The data in Table II shows that multi-camera distributed rendering reduces the amount of work load required by the main rendering gamelet and the net workload is approximately the same. While CPU data remains mostly the same in both cases and on the render assistant, the GPU usage decreases linearly as the number of pixels to render decreases. The marginal increase in CPU usage in cases 2,3 and 4 are attributed to the communication overhead with adjacent Gamelets.

TABLE II
AVERAGE CPU & GPU USAGE

Case No.	Size	**CPU**	**GPU**
1	Rendering full screen	13%	81%
2	Rendering half screen	14%	37%
3	Rendering half screen for another Gamelet	14%	38%
4	Rendering 1/6 of the screen	14%	26%

3) User Perception: To evaluate the user perception we conducted a small scale user study with seven undergraduate students. We set the network latency between game server and Gamelet to 120ms (above the acceptable latency of 100ms) and 200ms. Game server was run in a computer connected to our school's campus network, the Gamelets and clients were connected to two randomly selected WiFi access points of the campus wireless network. Users played the traditional client/server version of our custom developed 3D survival game called *Garden of Eden* for first 20 minutes to understand the game, learn the game mechanics and have a feel of the best possible version (non-cloud based). We assisted and trained them during this period.

After initial training, the users played different variants of the game to evaluate. There were five variants with different number of Gamelets to render for a client. The variants are presented to the users in random order. The client side code of the game simply gets the compressed stream from the Gamelet and uncompresses it to display. In addition, it captures all the user actions. We have developed *Laptop*, *iPad* and *Samrtphone (Android)* version of the client for evaluating. It is a multiplayer game. We used a mix of Laptops and iPads for each round in the user study. (A demo of the game with Gamelet concept is available in Youtube [26]).

The users were asked to look for artefacts (latency, jitter, visual quality, etc) if any and give a score for each version using 5-point Likert scale. The results are shown in Figure 11. Note: *Gamelets = zero* indicates pure cloud game (rendering is done in the game server itself). The Gamelet version of the game used 'dead reckoning' to hide latency. The pure cloud version assumes dump client. The client can only display the streamed images and capture user interactions. Initial results are encouraging. Gamelet system performs better than pure cloud based system especially when the delay is high. Hence, we claim Gamelets are good for multiplayer games over delay and loss prone wireless networks. However, when number of Gamelets are increased, the quality drops due to content synchronisation errors. We defer further work on improving synchronisation mechanism to future.

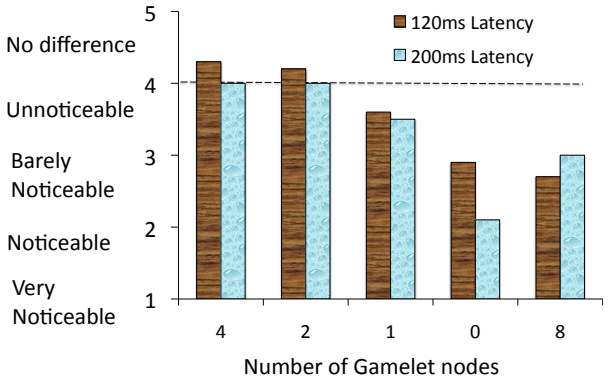

Fig. 11. User Study Scores

VI. DISCUSSION AND FUTURE WORK

Though the system works very well, there are two key limitations in distributed rendering part. The first limitation is the zone handling overhead. A constant additional computational load is added on the Gamelet due to the constant addition and removal of zones. This can be reduced by using appropriate zone size. We defer map and game genre based zone size optimisation to future work.

The second limitation is that the zones are downloaded with a predictive method. This means a user playing in the game world close to multiple boundaries will trigger the download of several zones that he might not require at all. In the worst

case scenario a user may be playing in an area with several overlapping boundaries, which in turn means he will download several additional zone data that fills up the Gamelet's memory for nothing. This may lead to the gamelet invoking peer assistance for rendering due to its inability to further download zone data. However, this case should only occur if the whole game consists of dense amounts of high polygon count models. Thus, good game programming practices can mitigate this limitation.

Our short user study is conducted with small group of student users in well controlled lab environments. We are planning to do a large scale user study with general public game players to study various other dynamics of the Gamelet system in detail.

Our proposal opens-up various *research challenges* including classical argument of distributed vs centralised control, synchronisation of Gamelets, compression techniques, security, Gamelet node trust, fairness of the game play (eg. connecting to powerful Gamelet vs average Gamelet node), mobility of players, dynamic zone resizing, energy efficiency [27] [28], overheads, resource provisioning and resource accountability. We plan to progressively address these challenges in our future works.

VII. Conclusion

In this paper we have made a *first attempt for a distributed micro-cloud infrastructure* with a next generation conceptual device called the *Gamelet*, which can be a separate device or integrated with the current wireless access point or a powerful game client to improve the performance and decrease the limitations faced by cloud game Industry. Key advantages over pure cloud games are: decreased latency, possibilities for latency hiding and synchronisation techniques, increased server scalability (esp. bandwidth). We strongly believe that Gamelet system and its variations will push the Cloud game Industry (especially, massively multiplayer mobile games) to next level. We sincerely thank Chong Ming Xun, Ngin Guan Wei Brain and Lee Tai Yun for their great help in implementing and testing the prototype game. Our special thanks to Yong U-Wern Justin for implementing content based adaptive streaming.

References

[1] H. T. Dinh, C. Lee, D. Niyato, and P. Wang, "A survey of mobile cloud computing: architecture, applications, and approaches," *Wireless Communications and Mobile Computing*, 2011. [Online]. Available: http://dx.doi.org/10.1002/wcm.1203

[2] N. Fernando, S. W. Loke, and W. Rahayu, "Mobile cloud computing: A survey," *Future Gener. Comput. Syst.*, vol. 29, no. 1, pp. 84–106, Jan. 2013. [Online]. Available: http://dx.doi.org/10.1016/j.future.2012.05.023

[3] C. Pettey, "Saas revenue to reach $14.5 billion in 2012," http://www.gartner.com/newsroom/id/1963815, Gartner, Mar. retrieved 2013.

[4] IDC, "Saas revenue to grow five times faster than traditional packaged software through 2014," http://www.idc.com/about/viewpressrelease.jsp?containerId=prUS22431810&sectionId=null&elementId, IDC, Jun. retrieved 2013.

[5] R. Shea, J. Liu, E.-H. Ngai, and Y. Cui, "Cloud gaming: architecture and performance," *Network, IEEE*, vol. 27, no. 4, pp. 16–21, 2013.

[6] L. Pantel and L. C. Wolf, "On the suitability of dead reckoning schemes for games," in *Proceedings of the NetGames*, ser. NetGames '02. New York, NY, USA: ACM, 2002, pp. 79–84. [Online]. Available: http://doi.acm.org/10.1145/566500.566512

[7] P. Sharkey, M. Ryan, and D. Roberts, "A local perception filter for distributed virtual environments," in *Virtual Reality Annual International Symposium, 1998. Proceedings., IEEE 1998*, 1998.

[8] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *Pervasive Computing, IEEE*, vol. 8, no. 4, pp. 14–23, 2009.

[9] K.-T. Chen, Y.-C. Chang, P.-H. Tseng, C.-Y. Huang, and C.-L. Lei, "Measuring the latency of cloud gaming systems," in *Proceedings of the 19th ACM international conference on Multimedia*, ser. MM '11. New York, NY, USA: ACM, 2011, pp. 1269–1272. [Online]. Available: http://doi.acm.org/10.1145/2072298.2071991

[10] D. Perry, "Why sony chose gaikai over onlive," http://www.gamestm.co.uk/discuss/why-sony-choose-gaikai-over-onlive/, gamestm, Jun. retrieved 2013.

[11] J. Cox, "Zynga grows to #1 social gaming site with rightscale," http://www.rightscale.com, rightscale, Jun. retrieved 2013.

[12] IDC, "The digipede framework sdk whitepaper," http://www.digipede.net/downloads/Digipede_SDK_Whitepaper.pdf, Digipede Technologies, Jun. retrieved 2013.

[13] J. Goodman and C. Verbrugge, "A peer auditing scheme for cheat elimination in mmogs," in *Proceedings of the ACM Netgames*, ser. NetGames '08. NY, USA: ACM, 2008.

[14] W.-c. Feng, E. Kaiser, and T. Schluessler, "Stealth measurements for cheat detection in on-line games," in *Proceedings of the Netgames*, ser. NetGames '08. NY, USA: ACM, 2008.

[15] H. K. Stensland, M. O. Myrseth, C. Griwodz, and P. Halvorsen, "Cheat detection processing: a gpu versus cpu comparison," in *Proceedings of the Netgames*, ser. NetGames '10. Piscataway, NJ, USA: IEEE Press, 2010.

[16] D. Rubinstein, I. Georgiev, B. Schug, and P. Slusallek, "RTSG: Ray Tracing for X3D via a Flexible Rendering Framework," in *Proceedings of the 14th International Conference on Web3D Technology 2009 (Web3D Symposium '09)*. New York, NY, USA: ACM, 2009, pp. 43–50.

[17] M. Lohse and M. Gmbh, "Network-integrated multimedia middleware, services, and applications," 2007.

[18] Unity, "Unity game engine," http://unity3d.com/, Unity Technologies, Jun. retrieved 2013.

[19] EpicGames, "Unreal game engine," http://www.unrealengine.com/, Epic Games, Inc., Jun. retrieved 2013.

[20] Rockstar, "Rockstar advanced game engine," www.rockstargames.com, RAGE Technology Group, Rockstar San Diego, Jun. retrieved 2013.

[21] CRYTEK, "CryENGINE," http://www.crytek.com/cryengine, CRYTEK, Jun. retrieved 2013.

[22] B. Anand, Z. Qiang, and A. L. Ananda, "Energy efficient multiplayer smartphone gaming using 3d spatial subdivisioning and pvs techniques," in *Proceedings of the 21th ACM International Conference on Multimedia, IMMPD*, Barcelona, Spain, Oct. 2013.

[23] B. Anand, K. Thirugnanam, L. T. Long, D. D. Pham, A. L. Ananda, R. K. Balan, and M. C. Chan, "Arivu: Power-aware middleware for multiplayer mobile games," in *Proceedings of the Ninth IEEE NetGames*, Teipei, Taiwan, Nov. 2010.

[24] A. Bhojan, A. Akhihebbal, M. Chan, and R. Balan, "Arivu: Making networked mobile games green," *Mobile Networks and Applications*, pp. 1–8, may 2011, 10.1007/s11036-011-0312-8. [Online]. Available: http://dx.doi.org/10.1007/s11036-011-0312-8

[25] B. Anand, A. L. Ananda, M. C. Chan, L. T. Long, and R. K. Balan, "Game action based power management for multiplayer online games," in *Proceedings of the 1st ACM SIGCOMM Workshop on Networking, Systems, and Applications on Mobile Handhelds (MobiHeld)*, Barcelona, Spain, Aug. 2009.

[26] *DEMO: Gamelets - Multiplayer Mobile Games with Distributed Micro-Clouds*, http://www.comp.nus.edu.sg/~bhojan/gamelets/index.html, National University of Singapore, 2013.

[27] K. Kumar and Y.-H. Lu, "Cloud computing for mobile users: Can offloading computation save energy?" *Computer*, vol. 43, no. 4, pp. 51–56, 2010.

[28] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "Maui: making smartphones last longer with code offload," in *MobiSys'10*, 2010.