Adaptive Multiple Texture Approach to Texture Packing for 3D Video Games

Alexander Wong
University of Waterloo
200 University Ave. West
Waterloo, Ontario
Canada
N2L 3G1
a28wong@engmail.uwaterloo.ca

Andrew Kennings
University of Waterloo
200 University Ave. West
Waterloo, Ontario
Canada
N2L 3G1
akenning@cheetah.vlsi.uwaterloo.ca

ABSTRACT

This paper presents an adaptive multiple texture approach to the problem of texture packing for 3D video games. In modern graphics hardware, texture size is typically constrained to width and height dimensions that are powers of two. To reduce the texture management overhead caused by storing individual textures, texture packing algorithms are used to pack multiple textures into a single powers-of-two texture. Current texture packing techniques are very limiting as they are capable of packing textures only into a single texture of predefined size. This can result in significant wasted texture space due to the powers-of-two texture size restrictions. In the proposed technique, individual arbitrarily sized rectangular textures are packed into multiple textures in an adaptive manner. This approach reduces the amount of wasted texture space in a more efficient manner by adaptively determining the quantity as well as size of textures being used during the packing process. Experimental results demonstrate the effectiveness of this technique in packing textures in an efficient and automated fashion. This makes it well suited for improving texture management in future 3D video games, where resources are limited and a high frame rate needs to be achieved to provide a truly immersive experience.

Categories and Subject Descriptors
I.3.7 [**Computer Graphics**]: Texture

General Terms
Algorithms

Keywords
texture packing, video games, 3D, adaptive

1. INTRODUCTION

A fundamental technique used in modern 3D video games is texture mapping [3]. In texture mapping, 2D images are mapped to the surface of 3D models to give them the appearance of greater detail and color. The main advantage of texture mapping is that it provides greater perceptual quality to 3D models without the increased computational complexity of using greater geometry for the models. This makes it well suited for real-time 3D applications such as video games, where a high frame rate needs to be achieved to provide a truly immersive experience.

The increase in graphical detail and visual realism in 3D video games have resulted in a dramatic increase in texture content both quantitatively and qualitatively. Therefore, it is important to keep the computational complexity given the large volume of texture data being used in a real-time 3D environment. The use of a large volume of individual textures results in performance penalties due to the need for an increased number of context switches in the graphics pipeline, as well as the performance cost of managing individual textures.

One common approach used to address the aforementioned problems is texture packing, where independent texture content are arranged in such a way that they can be stored in a larger texture. By packing individual textures into a larger texture, the number of context switches is significantly reduced as texture switches can be performed through texture coordinate offsetting. Furthermore, it is much less complex to manage a single texture than to manage a large number of individual textures. An example of texture packing is illustrated in Figure 1.

Many techniques have been proposed for the purpose of texture and and block packing [6, 5, 9, 11, 10, 8, 7]. We note that the texture packing problem is not a problem unique to graphics research. The problem of packing small rectangles into a single larger rectangle occurs, for example, in operations research and in VLSI physical design (e.g., fixed-outline floorplanning) [1, 2, 12]. Conventional texture packing techniques attempt to pack a set of individual textures into a single texture in such a way that the smallest possible area is utilized. A number of issues arise when these conventional texture packing techniques are used in the context of modern 3D video games. First, many of these algorithms

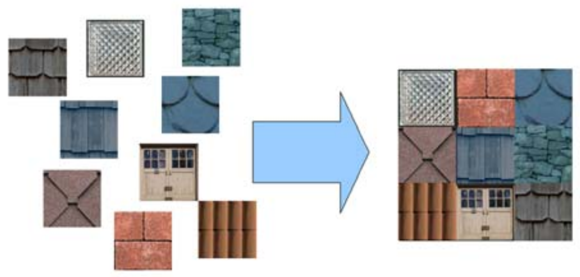

Figure 1: Example of a set of 9 individual textures packed into a single larger texture

assume that the initial size of the texture within which the individual textures are packed into is known a priori. Therefore, the texture artist must manually decide on the initial texture size, which can be difficult to do when a large number of textures are being packed into the single texture.

The second, and more serious issue, is that all current texture packing techniques assume that all textures must be packed into a single large texture. However, current graphics hardware have certain limitations when dealing with texture data that can make single texture packing very inefficient. First, current consumer graphics hardware are restricted to maximum texture sizes of 4096×4096. Therefore, conventional texture packing algorithms are unable to handle situations where the total area of the individual textures exceed the maximum texture size. Second, a majority of consumer graphics hardware only support textures with height and width dimensions that are powers of two (with only limited support of non-power-of-two textures in recent graphics hardware [4]). These dimension restrictions can result in significant wasted texture space when conventional texture packing algorithms are used.

For example, suppose that conventional texture packing algorithms are used to pack nine 128×128 textures. Since these techniques pack textures into a single larger texture, the valid texture sizes that will fit these textures while minimizing wasted space are 512×512, 1024×256, 256×1024, 2048×128, and 128×2048. Examples of texture packing using a 512×512 texture and a 1024×256 texture are illustrated in Figure 2. This results in 43.75% of texture space being wasted in the texture packing process. This is worsen by the fact that the absolute wasted texture space increases substantially as the powers of two increases. Therefore, a method that strikes a balance between the number of textures used for packing and the amount of wasted texture space based on the set of individual textures being packed is desired.

The main contribution of this paper is an efficient adaptive texture packing algorithm for packing individual textures into multiple larger textures in an automated fashion. Based on the set of textures that need to be packed, the proposed method automatically determines the number and size of textures used for packing such that the amount of texture

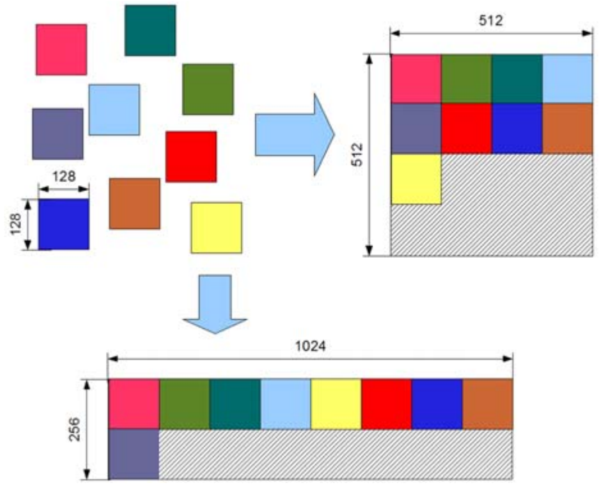

Figure 2: Example of packing nine 128×128 textures into a single 512×512 texture and a single 1024×256 texture. The hashed area represents wasted texture space

space being wasted is kept as low as possible while reducing the number of textures used. This allows for more efficient texture management in future 3D video games while reducing resource requirements, thus allowing for an improved game experience.

In this paper, the proposed texture packing method is presented in Section 2. A detailed analysis of the texture packing performance of the proposed method is presented in Section 3. Finally, conclusions are drawn and future work is discussed in Section 4.

2. PROPOSED ADAPTIVE TEXTURE PACKING ALGORITHM

In the proposed method, a set of n input textures $\{T_1, T_2, ..., T_n\}$ is packed into a set of m output textures $\{O_1, O_2, ..., O_m\}$, where $n > m$. Each output texture has the following restrictions:

- The width and height of the texture are powers of two, and
- The width and height of the texture cannot exceed the maximum texture dimensions.

The number and size of output textures are not known a priori. Therefore, the set of output textures will vary based on the input textures such that a balance between the number of textures and the amount of wasted texture space. The main advantage of the proposed method is that the amount of wasted texture space can be substantially reduced while keeping the number of textures being managed as low as possible. An overview of the proposed texture packing method is illustrated in Figure 3. The set of input textures are analyzed and an initial guess for the size and number of output textures is made. The set of input textures are then packed based on the initial guess and analyzed to further adjust the number and size of output textures to reduce wasted texture space and produce the final set of output textures.

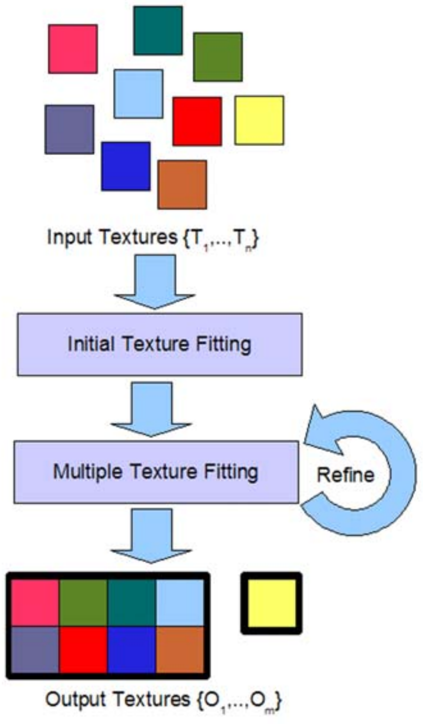
Input Textures $\{T_1,...,T_n\}$
Initial Texture Fitting
Multiple Texture Fitting
Refine
Output Textures $\{O_1,...,O_m\}$

Figure 3: Overview of the proposed texture packing algorithm

2.1 Initial Texture Packing

In the initial texture packing stage, it is necessary to determine an initial estimate for the size and number of output textures needed to pack the set of n input textures $\{T_1, T_2, ..., T_n\}$ during the multiple texture packing stage. The dimension for each texture in the set of input textures is known a priori. Let (w_i, h_i) be the width and height dimensions of input texture T_i. As a first step to the estimation process, the total texture area covered the set of input textures is computed as follows:

$$A = \sum_{i=1}^{n} w_i h_i \tag{1}$$

Suppose we wish to pack the entire set of input textures into a single square texture F. Since the width and height of a texture must be powers of two, the texture F would have the dimensions $(w_F, h_F) = (2^j, 2^j)$, where j is a parameter that controls the size of the texture. Therefore, the following condition must hold true if the set of input textures were to fit into the texture F:

$$A \leq 2^{2j} \tag{2}$$

Based on the above condition, a good initial estimate for parameter j would be the following:

$$j = \left\lceil \frac{\log_2(A)}{2} \right\rceil \tag{3}$$

We note that j in Equation (3) is limited by the graphical hardware limitations for texture size. In this case we restrict j in which case the algorithm will obviously fail to produce a solution if only a single output texture is used. However, our algorithm handles this case as an 'over-fitted' scenario which is described later in the paper.

To determine an initial estimate of the number and size of the output textures within which the input textures are packed, the set of input textures are first packed purposefully into the square texture F so that the texture area usage can be analyzed. In the proposed algorithm, a texture packing system based on the whitespace management concepts presented in [12] and [1] was utilized. White space management is a well-known technique that has been widely used in user interfaces for the purpose of 2D and 3D layout. However, such a technique has not been utilized for the purpose of automatic multiple texture packing, which has its own set of requirements. The proposed texture packing system allows multiple arbitrarily sized rectangular textures to be packed into a texture area while accounting for the restrictions imposed by current graphics hardware. The proposed texture packing system works as follows:

1. The set of input textures $\{T_1, T_2, ..., T_n\}$ are sorted in decreasing order based on their texture area (i.e., from largest to smallest).
2. The whitespace list is initialized with a single whitespace rectangle with the same dimensions as texture F.
3. The largest unfitted input texture is packed into texture F and the whitespace list is modified by replacing the single whitespace rectangle with adjacent whitespace rectangles formed around the packed texture.
4. For the next largest unfitted input texture, the whitespace list is checked to find the smallest whitespace rectangle that can hold the input texture. If no appropriate whitespace rectangle is found, the algorithm is terminated and texture F is classified as 'over-fitted'. Otherwise, the input texture is packed into the appropriate whitespace partition within texture F. If all input textures have been packed into texture F, the algorithm is terminated and texture F is classified as 'fitted'.
5. The whitespace list is modified by recalculating the largest available rectangles within texture F and the algorithm proceeds back to Step 4.

A example of the proposed texture packing system is shown in Figure 4.

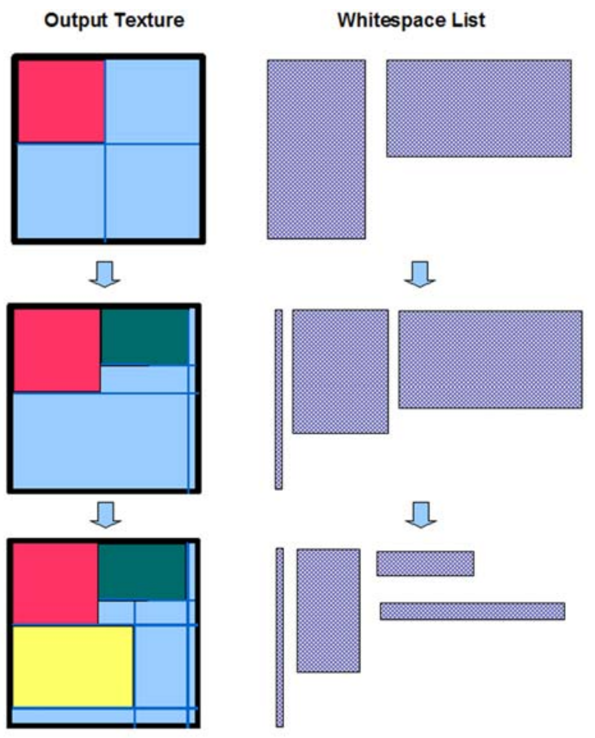

Figure 4: Example of the proposed texture packing system

2.1.1 Fitted Scenario

If the texture F was classified as 'fitted', the percentage of wasted texture space, W, is calculated as follows:

$$W = \frac{A}{2^{2j}} \quad (4)$$

Once the percentage of wasted texture space has been calculated, it is necessary to determine whether a single texture solution or a multiple texture solution provides a better fit for the current set of input textures. To determine whether a multiple texture solution is necessary, a check is performed to see if either of the following conditions are met:

1. The area covered by an individual input texture is greater than 25% of texture F.
2. The total area covered by the set of input textures is greater than 75% of the texture F.

If either of the conditions hold true, then the packed texture F is set as the final output texture O_1, as the single texture fitting provides a very good fit for these scenarios and no further output textures are needed. If the conditions are not met, an initial estimate utilizing multiple textures is desired.

The initial estimate utilizing multiple textures was derived in the following manner. First, by failing to satisfy the aforementioned conditions, it is reasonable to make the assumption that the entire set of input textures can be packed into three quarters of the square texture F. However, due to the fact that the dimensions of the output textures must be powers of two, it is not possible to construct a rectangular output texture with an area equal to $3(2^{2j})/4$. One approach to addressing this problem is to subdivide the square texture into four $2^{j/2} \times 2^{j/2}$ square textures, thus retaining dimensions that are powers of two. Since only three of the smaller textures are needed, the fourth texture can be discarded. This approach leads to a texture space savings of 25% when compared to that used by the texture F. However, this approach also results in three times as many output textures. To reduce the number of output textures required, two of the remaining output textures can be merged into a $2^j \times 2^{j/2}$ rectangular texture. This reduces the number of output textures from three to two.

Based on the above reasoning, in the event that a multiple texture estimate is desired for the under-fitted case, the initial estimated set of output textures consists of the following two textures:

1. An output texture (O_1) with dimensions $(w_1, h_1) = (2^j, 2^{j/2})$.
2. An output texture (O_2) with dimensions $(w_2, h_2) = (2^{j/2}, 2^{j/2})$.

2.1.2 Over-fitted Scenario

If the texture F was classified as 'over-fitted', the initial estimates for the output textures can be derived based the following logic. As previously mentioned, the square texture F was constructed such that its area is greater than or equal to the total area covered by the set of input textures. Therefore, the fact that the texture F cannot hold the entire set of input textures implies that the dimensions of the input textures are not powers of two. Furthermore, the total area of the remaining input textures that did not fit into F cannot exceed $2^{2j} - A_{packed}$, where A_{packed} is the total area of all input textures that can be packed into F. In the situation where the area covered by an individual input texture is greater than 25% of F, then it is not possible to fit more than one texture within F. As such, a very good fit for this situation is a single output texture with dimensions $(w_1, h_1) = (2^{2j}, 2^j)$. In the situation where the area covered by an individual input texture is less than 25% of F, the remaining textures is likely to fit within an output texture with dimensions $(w_2, h_2) = (2^{j/2}, 2^{j/2})$, which has an area that is a quarter of texture F.

Based on the above reasoning, the initial estimates for the over-fitted scenario can be determined in the following manner. First, a check is performed to see if the area covered by an individual input texture is less than 25% of the texture F. If the area covered by an individual input texture is less than 25% of the texture F, the initial estimated set of output textures consists of the following two textures:

1. An output texture (O_1) with dimensions $(w_1, h_1) = (2^j, 2^j)$.

2. An output texture (O_2) with dimensions $(w_2, h_2) = (2^{j/2}, 2^{j/2})$.

If the area covered by an individual input texture is greater than 25% of the texture F, the initial estimated set of output textures consists of one output texture (O_1) with dimensions $(w_1, h_1) = (2^{2j}, 2^j)$.

2.2 Multiple Texture Packing

Once the initial estimates for the number and size of output textures has been established, texture packing is performed on the set of initial output textures in the following manner:

1. The texture packing system described in Section 2.1 is used to pack the set of input textures into output texture O_1.
2. The remaining textures that cannot be packed into O_1 are packed into O_2 in situations where multiple output textures exist.

Once the initial multiple texture packing has been performed, an analysis of texture space usage is performed on O_2 to further refine the size and number of output textures used.

2.2.1 Fitted Scenario

For the cases where the texture F was classified as 'fitted', the size and number of output textures can often be refined to obtain a better fit for the set of input textures. The main obstacle to reducing the texture space needed to store the set of input textures is the fact that the output textures must maintain power-of-two dimensions. Therefore, refinements in output texture sizes must satisfy the restrictions in texture dimensions to function properly on the majority of consumer graphics hardware. The approach taken by the proposed texture packing method is to decrease the area of output textures by half in alternating dimensions until the minimum power-of-two texture size is found.

Based on the above reasoning, the following texture refinement process is performed:

1. Check if the texture usage of O_2 is zero. If the texture usage is zero, set O_1 to size $(w_1/2, h_1)$ and set O_2 to size $(w_1/2, h_1/2)$ and repack. If all textures fit within these two textures, terminate the algorithm. Otherwise, set O_1 back to its original size, remove O_2, and terminate the algorithm.
2. Check if the textures that cannot be packed into O_1 can be packed into O_2. If the textures cannot be packed, remove O_2 and set F as O_1 and terminate the algorithm. Otherwise, divide the height of O_2 by half, repack, and proceed to Step 2.
3. Check if the refined O_2 can hold all remaining textures. If the textures cannot be packed, revert to the previous size of O_2, repack, and terminate the algorithm. Otherwise, divide the width of O_2 by half, repack, and proceed to Step 3.
4. Check if the refined O_2 can hold all remaining textures. If the textures cannot be packed, revert to the previous size of O_2, repack, and terminate the algorithm. Otherwise, divide the height of O_2 by half, repack, and proceed to Step 2.

An example of this refinement process is shown in Figure 5.

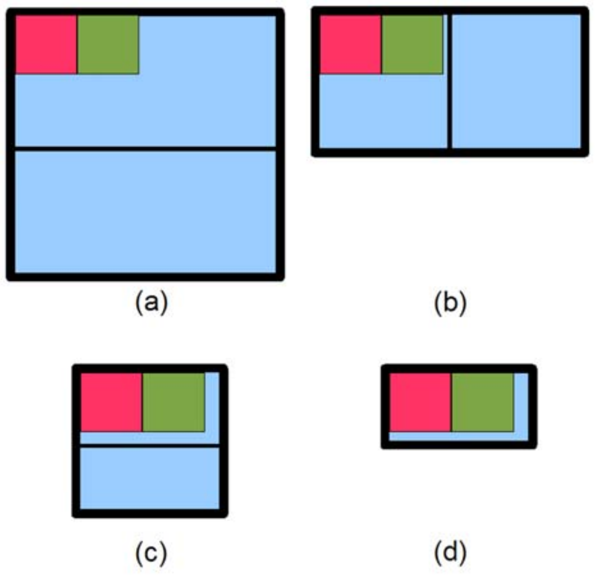

Figure 5: Example of the refinement process for the fitted scenario. The texture is reduced by half in alternating dimensions until the minimize size is achieved.

2.2.2 Over-fitted Scenario

For the cases where the texture F was classified as 'over-fitted', the texture refinement process is refined iteratively in the following manner. First, a check is performed to see if all input textures can be packed into the current set of output textures. If the input textures cannot be packed, the width and height of the smallest output texture is doubled in an alternating manner until either:

- All input textures can be packed into the refined set of output textures, or
- the size of the smallest output texture is equal to the size of the largest output texture.

If all the input textures can be packed into the refined set of output textures, then a fit is found and the resulting set of output textures are used as the final set of output textures. If the size of the smallest output texture is equal to the size of the largest output texture and a fit is still not found, a new output texture with a size equal to the smallest output texture prior to refinement is added to the set of output textures and the refinement process is performed repeatedly until all the input textures can be packed into the set of output textures. Once the final number and size of output textures are determine, the input textures are packed into the set of final output textures.

3. TEXTURE PACKING PERFORMANCE ANALYSIS

To demonstrate the effectiveness of the proposed adaptive texture packing method, texture packing was performed on four test sets of generated input textures. A brief description of each test set is provided below.

TEST 1: Set of 133 [32 × 32] textures.

TEST 2: Set of 31 [50 × 50] textures.

TEST 3: Set of 149 [64 × 64] textures.

TEST 4: Set of 10 [300 × 300] textures.

To judge the effectiveness of the proposed method quantitatively, the percentage of wasted texture space was determined. For comparison purposes, the test input textures in each test set was packed into the smallest possible power-of-two texture.

The percentage of wasted texture space for each scenario is shown in Table 1. It can be observed that the amount of wasted texture space was significantly reduced using the proposed method when compared to the use of a single texture. Examples of the resulting packed output textures are shown in Figure 6, Figure 7, and Figure 8. This demonstrates the effectiveness of the proposed texture packing method in providing a balance between the amount of textures and the amount of texture space used.

4. CONCLUSIONS AND FUTURE WORK

This paper proposes a novel adaptive texture packing algorithm that utilizes multiple textures in an automated manner. The proposed method adapts the size and number of output textures based on the set of arbitrarily sized input textures to reduce the amount of wasted texture space while maintaining a small number of textures. It is believed that the proposed technique can be used effectively for the purpose of texture packing to improve texture management in future 3D video games. Future work include investigating alternative packing methods to further improve texture packing efficiency.

5. REFERENCES

[1] B. Bell and S. Feiner. Dynamic space management for user interfaces. In *User Interface Software*, pages 239–248, 2000.

[2] M. Bernard and F. Jacquenet. Free space modeling for placing rectangles without overlapping. *Journal of Universal Computer Science*, 3(6):703–720, 1997.

[3] E. Catmull. Computer display of curved surfaces. In *IEEE Conference on Computer Graphics, Pattern Recognition, and Data Structures*, pages 11–17, 1975.

[4] M. Corporation. *DirectX 9 reference manual*, chapter D3DPTEXTURECAPS_NONPOW2CONDITIONAL. Microsoft Corporation, 2003.

[5] P. Decaudin and F. Neyret. Packing square tiles into one texture. In *Eurographics 2004*, pages 49–52, August 2004.

[6] P. Erdos and R. Graham. On packing squares with equal squares. *Journal of Combinatoral Theory (A)*, 19:119–123, 1975.

[7] E. Friedman. Packing unit squares in squares: a survey and new results. *The Electronic Journal of Combinatorics*, DS7:1–24, 1998.

[8] D. Jennings. On packings of squares and rectangles. *Discrete Mathematics*, 138(1):293–300, 1995.

[9] K. Kaul and C. Bohn. A genetic texture packing algorithm on a graphical processing unit. In *The Ninth International Conference on Computer Graphics and Artificial Intelligence*, 2006.

[10] W. Li and A. Kaufman. Texture partitioning and packing for accelerating texture-based volume rendering. In *Graphics Interface 2003*, pages 81–88, 2003.

[11] H. Nagamochi. Packing unit squares in a rectangle. *The Electronic Journal of Combinatorics*, R37:1–13, 2005.

[12] K. Vorwerk, A. Kennings, D. Chen, and L. Behjat. Floorplan repair using dynamic whitespace management. In *Journal of Universal Computer Science*, pages 552–557, 2007.

Table 1: Percentage of wasted textures space for test scenarios

Scenario	Input Textures (Quantity/Size)	Output Textures (Quantity/Size)	Wasted Texture Space Using Proposed Method	Wasted Texture Space Using Single Texture
TEST 1	$133/(32 \times 32)$	$1/(512 \times 256)$ $1/(128 \times 64)$	2.26%	48.05%
TEST 2	$31/(50 \times 50)$	$1/(256 \times 256)$ $1/(256 \times 128)$	21.16%	40.87%
TEST 3	$149/(64 \times 64)$	$1/(1024 \times 512)$ $1/(512 \times 256)$	6.88%	41.80%
TEST 4	$10/(300 \times 300)$	$1/(1024 \times 1024)$ $1/(512 \times 512)$	31.34%	57.08%

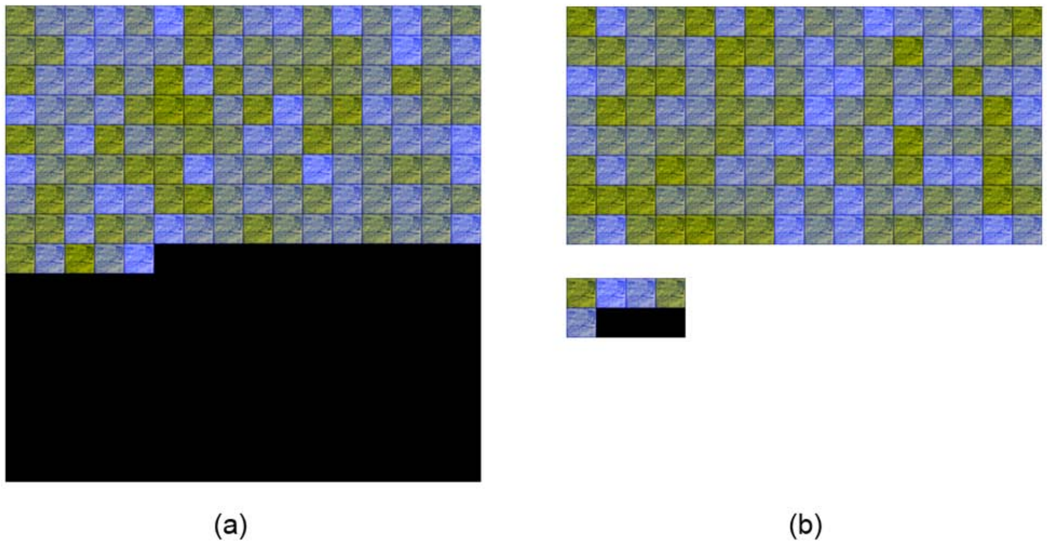

Figure 6: TEST 1: a) Texture packing using single texture, b) Texture packing using proposed method

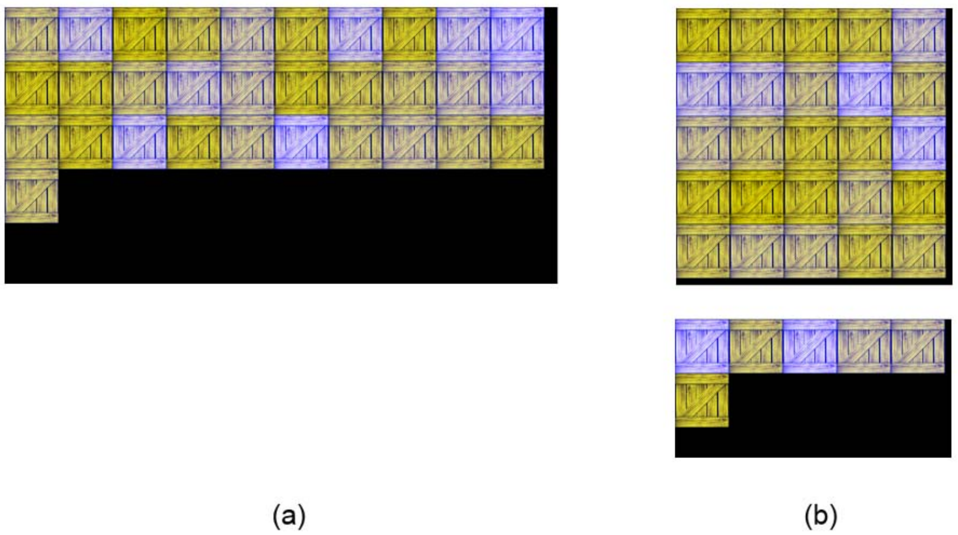

Figure 7: TEST 2: a) Texture packing using single texture, b) Texture packing using proposed method

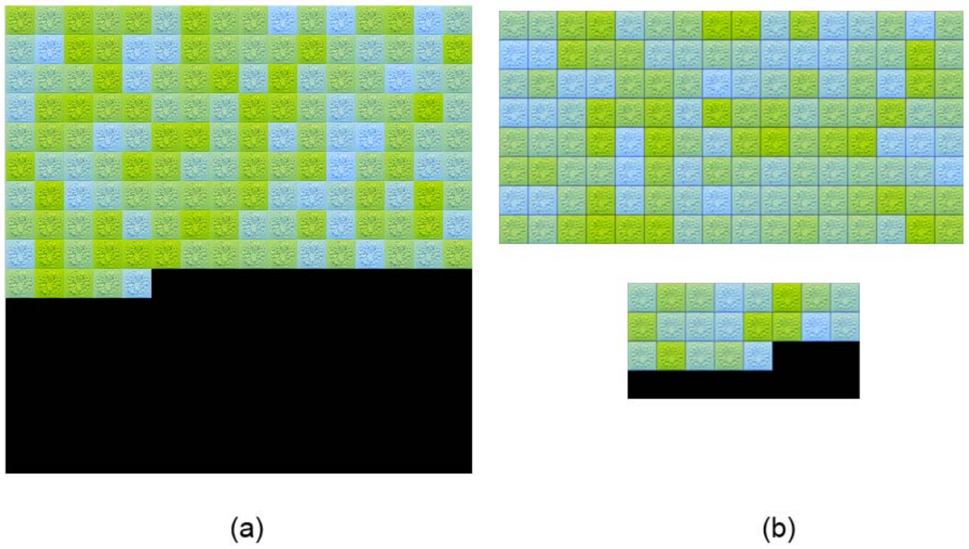

Figure 8: TEST 3: a) Texture packing using single texture, b) Texture packing using proposed method