Texture Packing

张天涵 国坤晨 李更新

Date：2022.5.16

目录

一、项目简介

1.1问题描述

这个问题可以表述如下,给定一个给定宽度 W 和无限长度的矩形板，和一组 n 个矩形，每个矩形 i 的长度为 Li和宽度 Wi，在板使得没有两个矩形重叠并且使用过的板长度 H 被最小化。

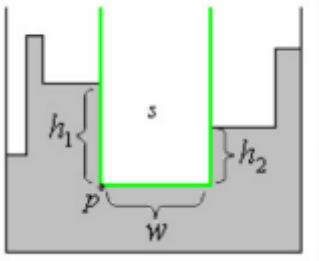
其中，矩形设定为不可旋转。

二、数据结构/主要算法

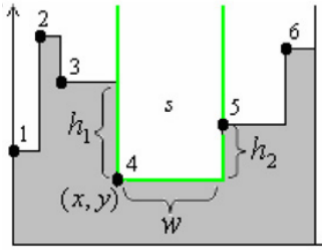
2.1相关信息说明

2.1.1 Space

在讲算法之前，我们先给“空间”一个定义：空间指堆叠体顶部的线段，到顶端的竖直区域，如图中绿色区域所示：



整个区域的放置可能位置是由一系列的“空间”组合而成，我们依据它们的定位点来进行排序联系在一起，如图所示：



（图中1、2、3...6分别代表对应的空间，顺序从左往右，整个空白区域被划分成了6份）

具体结构的实现请参考代码。

对应参数表：

参数名	含义
h1,h2	左墙与右墙高
p	空间定位点
s	空间标识

注意：我们在具体设计时，将参数p换成了x,y，用来表示p的空间坐标位置。

其中，给定区域的最左下角坐标为（0，0）。

2.1.2 Rec

矩形参数表

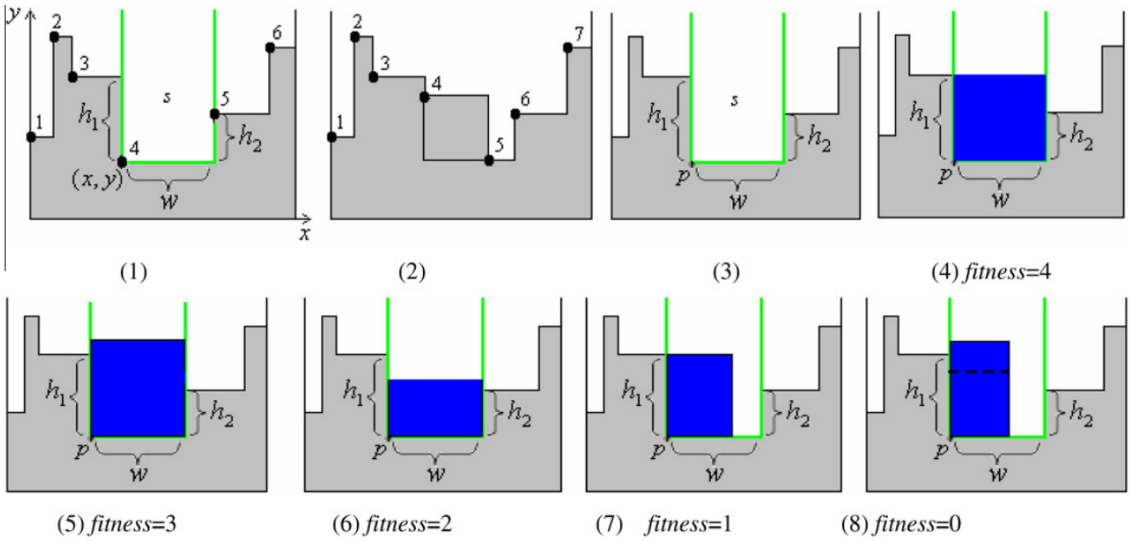
参数名	含义
x,y	矩形左下角顶点坐标
width,length	矩形宽度，高度

2.2所用算法

2.2.1 方案设计

采用赋分制：

我们向这个矩形框里放入矩形时分为以下几种情况，并给这几种情况（4~8）进行不同的赋分（满分为4分）



矩形得分越高，说明矩形与当前空间越匹配，在当前放置平面放置的优先级就越高，我们选取得分最高的矩形放在固定位置。

矩形的得分策略如下

	Cases	Score	Space Change(s)
	w = r[i].width and (h1 or h2) = r[i].length	4	-2 or -1
	w = r[i].width and max(h1,h2) < r[i].length	3	0
	w = r[i].width and max(h1,h2) > r[i].length	2	0
	w > r[i].width and (h1 or h2) = r[i].length	1	0
	w > r[i].width and (h1 and h2) ≠ r[i].length	0	+1

不同得分下，空间集合S及被选空间s的变化：

得分	修改内容
4	如果h1 = h2，说明完全填充，空间数减少2，填充空间边长w_s变为 $w_s + w_{left} + w_{right}$ ，h1、h2随之更新，然后去除两个相邻空间s_{left}和s_{right}；如果h1 ≠ h2,说明部分填充，与合适的齐平，空间数减少1，填充空间边长w_s变为与合适空间的边长和，即：$w_s + w_{proper}$ 。h1、h2随之更新，然后去除同高的空间
3	修改h1、h2成无穷就好，然后更新$s_{left}.h2$ 和$s_{right}.h1$即可
2	修改h1、h2成相对差值，然后看情况更新相邻空间高度（两种情况，比矮墙空间高或者低）
1	修改当前空间的w,同时修改对应匹配的空间的w就好
0	增加一个新空间，具体需要根据h1与h2的大小关系确定，然后修改h1和h2

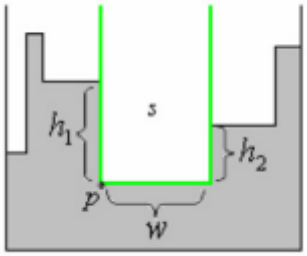
下面是具体案例：

2.2.2实例说明

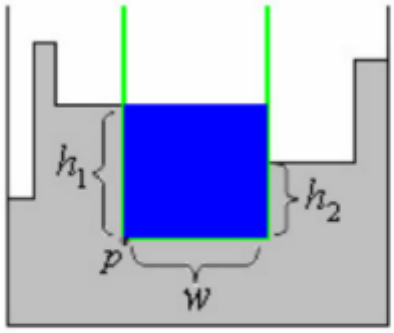
放置第一个矩形时， 空间左右墙都为无穷高，我们把这个状态视作初始状态，直接选取最大的矩形放入即可。然后接下来放入矩形分为5个优先级，每次放入都要遍历一遍，找出优先级最高的矩形放入，放入之后更新空间信息。下面分越高优先级越高

放完矩形更新空间之后选择插入矩形的空间符合最低最左原则，即先找到最低的边，如果有多条边的纵坐标相同，则选取其中最靠右的边作为将要插入矩形的空间。

这是插入前，这个矩形框中的状态



满分4分匹配时的空间变化：



(4) *fitness*=4

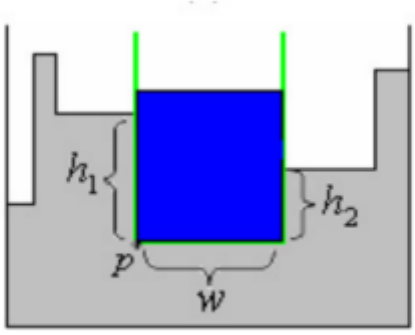
特点：宽度完全填充，不存在左右靠齐问题

先判断空间左右高度是否齐平

若齐平：删除两个相邻空间，修改w，h1,h2

若不齐平：找到与矩形高度相符合的边，删除该边相邻的空间，修改当前空间的w,h1,h2 即可

3分匹配时的空间变化：

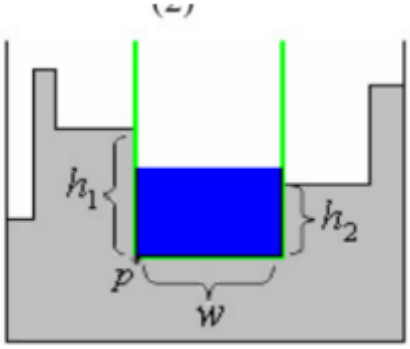


(5) *fitness*=3

特征：宽度正好匹配，但是矩形长度 length 比空间两墙都高（仅考虑有限高墙体，无穷高墙体高度视作-1）

当前空间S起始坐标y增加h，两侧墙体高度h1,h2变为无穷

2分匹配时的空间变化：



(6) *fitness*=2

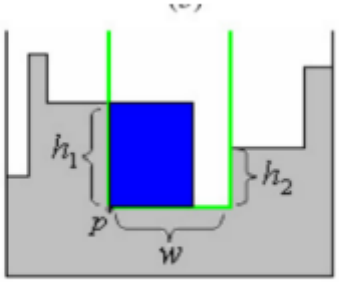
特征：宽度正好匹配，但是矩形高度比较高墙低，这个时候需要分为三种子情形：

高度与低墙相等（这个归属于4分匹配）： 当前空间S坐标y增加h，高墙高度下降h，低 墙高度与相邻空间的另一侧墙同高，w增加，同时删除相邻墙体；

高度大于低墙： 当前空间S坐标y增加h, 高墙高度下降h，低墙高度变为无穷，同时相邻空 间墙体高度由无穷变为h-S的原低墙高 ；

高度小于低墙： 当前空间S坐标y增加h, 高墙高度下降h，低墙高度下降h；

1分匹配时的空间变化 :



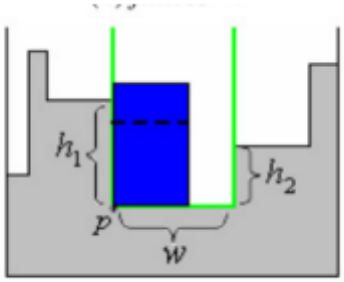
(7) *fitness*=1

特征：宽度不同，但与某一墙体等高，分为两种子情形：

靠左墙：空间变化如白色部分所示，前空间宽度增加，当前空间宽度减小，坐标x增加width

靠右墙：变化同上，仅对象改变，后空间宽度增加，当前空间宽度减小，坐标x不变

0 分匹配时的空间变化



3) *fitness*=0

特征：无特征，完全没有匹配

我们选择将矩形靠近较高一边的墙体放置

2.3关键设计的伪代码说明

伪代码：HeuristicPacking()总体的流程框架

```
h:=0;pin:=0;     {h为总高度，pin为已经打包好的矩形数量}
while pin<n do  {n为矩形总数}
    find the lowest and the most left space s;  {F1:LL_SpaceChoose}
    if S[s].w >= minwidth then  {如果当前空间宽度比待pack的矩形最小宽度大，说明肯定能够放入一个矩形}
        for each unpacked rectangle i do
            give a score to r[i];   {F2:Score}
        select a rectangle R with the maximum score from unpacked rectangles;{这里默认为第一个达到最高分的矩形}
        pin:= pin + 1;
        if S[s].y+r[R].length > h then  {更新总高度判断}
            h:= S[s].y + r[R].length;
        update space set S; {F3:UpdateSpaceSet}

        if r[R].width = minwidth then
```

```
            Renew the minwidth; {F4:FreshMinWidth}
    else {否则就将不满足条件的缺口填平，修改空间集合的信息}
        update space set S;
return h;
```

F2伪代码:Score()

```
if r.width == s.w then  {这里r为选定矩形，s为选定空间}
    if length = s.h1(or s.h2) then
        return 4; {The full mark}
    if length > max(h1,h2) then {这里我们设定无穷大为-1，故初始插入的最优情况为3分}
        return 3;
    else return 2;
else if r.width< s.w then
    if length = s.h1(or s.h2) then
        return 1;
    else return 0;
else then
    Error report;
    return __NOT_FIT__;
```

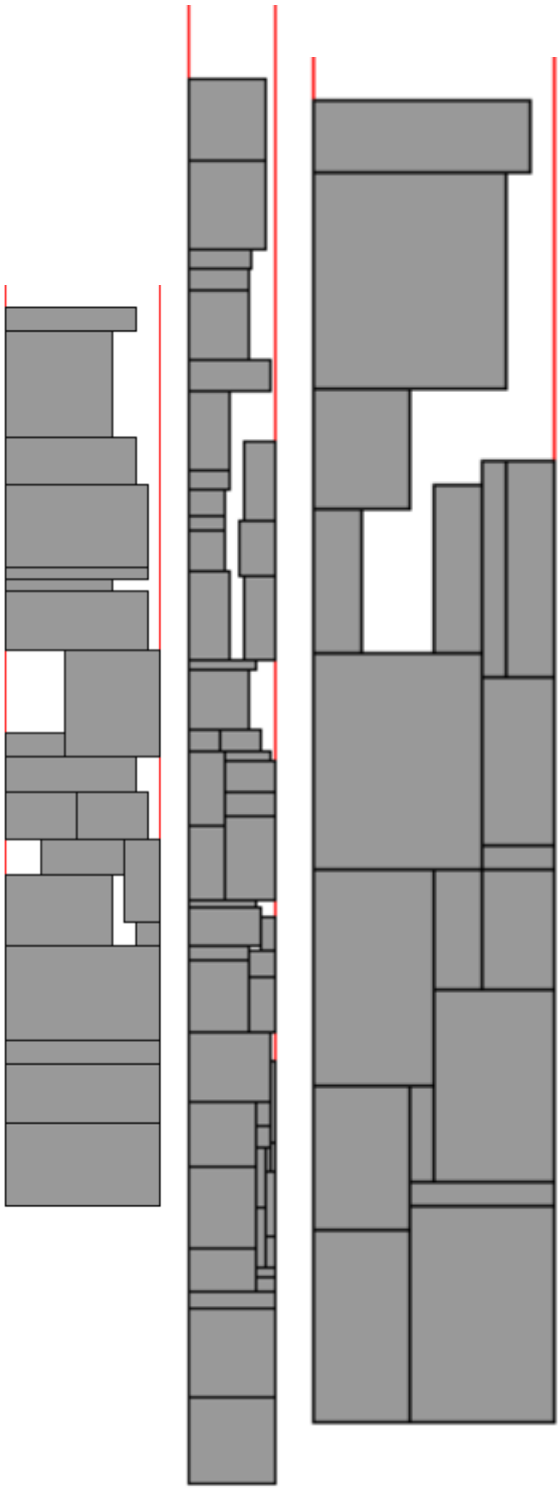
F3伪代码：UpdateSpaceSet()

详情流程请参考算法设计环节以及具体代码

```
switch(score)
    case 4:
        Pack rectangle; {F5:PackToLeft/PackToRight}
        merge Space(s) and remove redundant space;
        break;

    case 3/2/1:
        Pack rectangle;
        modify current space infomation with proper method;
        break;

    case 0:
        Pack rectangle;
        modify infomation and insert a new space;
        break;

    default:    {这里为无法放入矩形的情况，程序中为__NOT_FIT__,值为-1}
        fill and merge the current space,and remove redundant space.
```

三、测试结果样例

仅给出部分测试结果参考图，更多测试结果的具体说明在后面的附录中



四、分析

时间复杂度(这里的时间复杂度分析不包含优化方案部分)

$O(n^2), n$为矩形的个数

其中在单次放置中，对矩形进行赋分和选择操作需要$O(n)$复杂度的时间，放置需要$O(1)$的时间，总共至少需要n次放置，故时间复杂度粗略估计为：

$$O(n^2) \tag{1}$$

空间复杂度

$O(n)$，n为矩形的个数

为了存储所有矩形信息，我们需要$O(n)$复杂度的空间，而存储“空间”信息，至多需要 n 个空间（这一情况出现在所有矩形都堆积在最底部），其余的变量空间都占用常数数目的空间，故空间复杂度为：

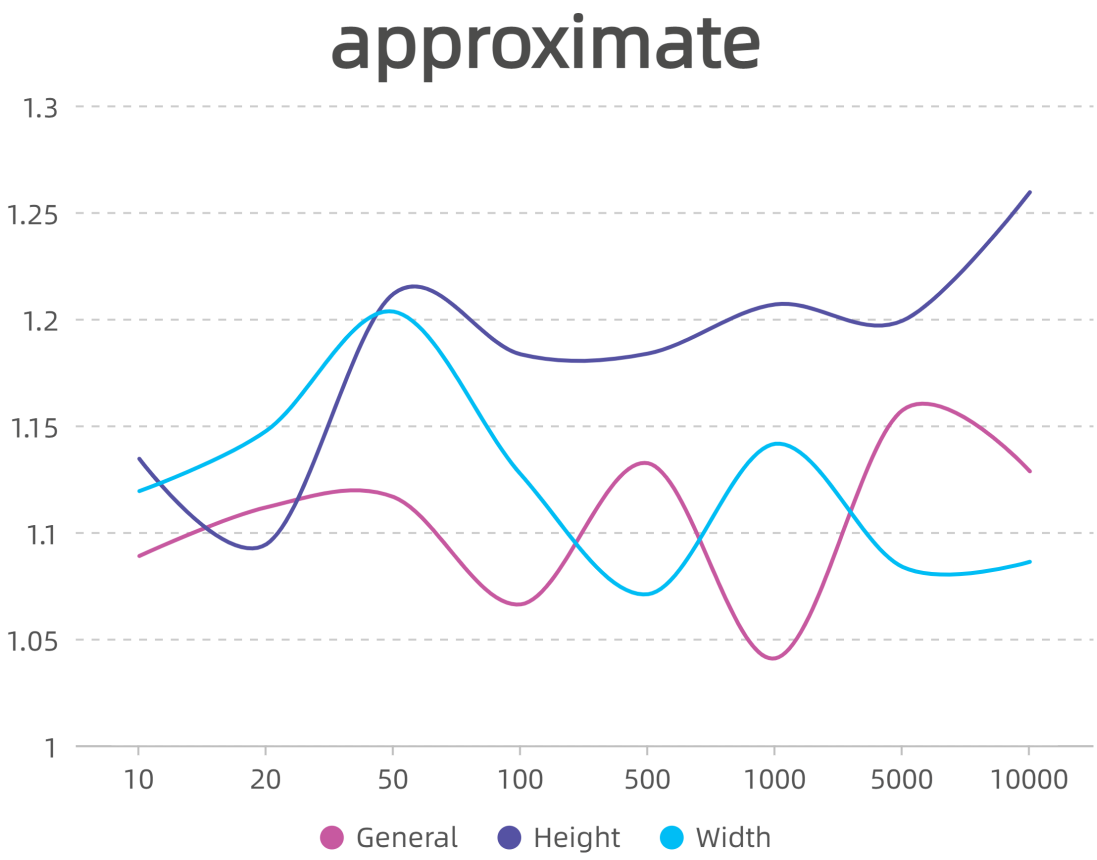
$$O(n) \tag{2}$$

近似度分析

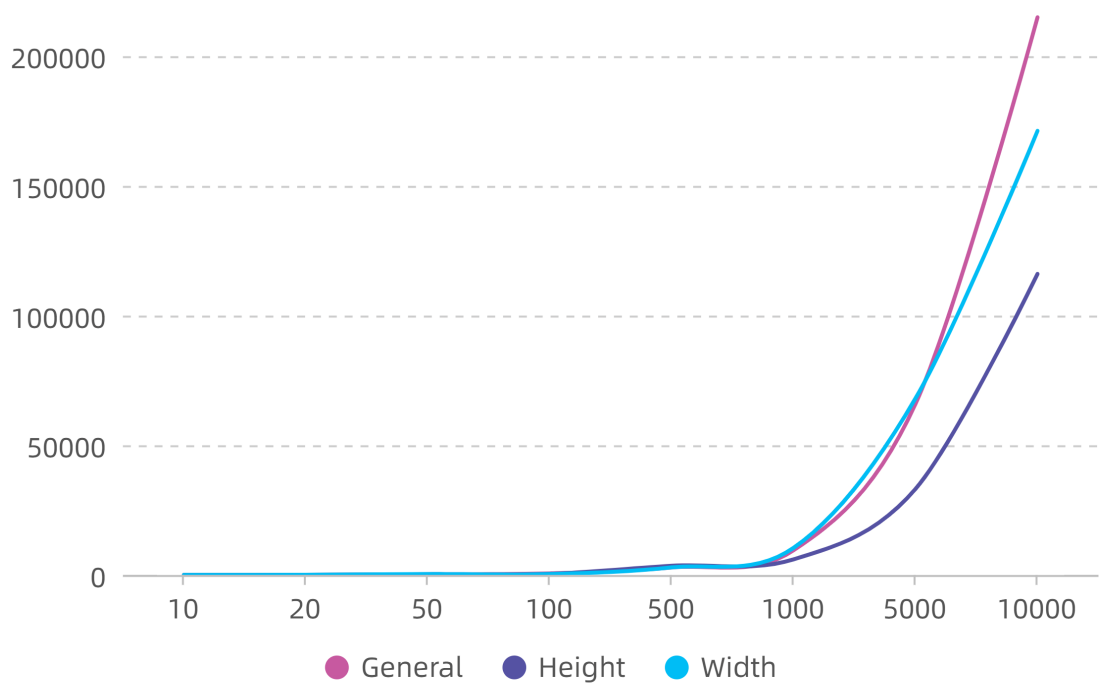
这里我们采用下述公式来计算：

$$\text{近似比} = \frac{\text{区域总面积}}{\text{矩形总面积}} \tag{3}$$

同时我们采用折线图来显示多个测试样例下的结果：



对应的当前算法下的最佳高度：



Tips:

以上测试用例的数据我们放在对应的文件夹中，使用的话需要将内容复制到 TestCase 文件夹的 data.txt 文件中，程序才能正常读取内容；同时会通过控制台显示最小高度，并将所有矩形的位置信息输出到 result.txt 文件中

五、参考优化方案

根据Alvarez-Valdes等人2008年发表的论文（Reactive GRASP for the strip-packing problem. Computers and Operations Research 35, 1065–1083）中的分析，可以获知放置结果与矩形的放置顺序相联系，至此我们可以从这个角度对放置方案进行优化。

即**通过修改矩形排列的序列，重复放置流程，来尽量避免所得结果为局部最优解。**

优化的参考方案:

在第一次获取高度后，选择序列中的两个矩形交换序列位置，用新的序列重新进行高度获取，比较新结果与原结果，如果更好，就更新最优解，同时更新序列为新序列。重复上述操作数次，就可以尽量减小所得结果为局部最优解的可能。

即是用Local Search方法跳出局部最优解

下面给出参考伪代码：Optimization()

```
sort all unpacked rectangles by non-increasing ordering of width and obtain ordering X;
besth:= HeuristicPacking(X),bestX:=X;
for i:=1 to n-1 do
    for j:= i+1 to n do
        swap the order of rectangles i and j in X and obtain a new ordering M;
        current_h:=HeuristicPacking(M);
        if current_h < besth then
            besth:=current_h;
            bestX:=M , X:=M;
return besth and bestX;
```

附录：

1.测试结果

	10G	10H	10W	20G	20H	20W	50G	50H
Width	8	8	9	17	10	13	19	21
Min height	23	19	24	75	58	76	305	282
approximater	1.0887	1.1343	1.1191	1.1116	1.0943	1.1475	1.1116	1.2117

	50W	100G	100H	100W	500G	500H	500W	1000G	1000H
Width	26	23	21	18	24	38	26	36	37
Min height	341	606	347	442	2896	3546	2916	9543	6024
approximater	1.2033	1.0661	1.1833	1.1269	1.1322	1.1837	1.0709	1.0408	1.2067

	1000w	5000G	5000H	5000W	10000G	10000H	10000W
Width	47	60	46	64	94	83	82
Min height	10456	66007	33203	68129	214987	116091	171212
approximater	1.1414	1.1572	1.1990	1.0836	1.1284	1.2593	1.0860

测试样例说明：测试样例名称由两部分构成，第一部分的数字N表示方块数量，第二部分的字母有"G""W""H"三种情况，其中：

G：表示general 普通的方块构成， 宽度和高度没有明确的数量关系
W：表示width 宽度大于高度的方块居多，即又矮又宽的方块数量为又高又瘦的方块数量的四倍左右
H：表示height 高度大于宽度的方块居多，即又高又瘦的方块数量为又矮又宽的方块数量的四倍左右

测试样例格式： （第一行给出空间宽度以及矩形总数N，之后的2到N+1行给出具体N个矩形的宽度和高度，以空格分隔。

空间宽度，矩形总数
第一个矩形的宽度 高度
第二个矩形的宽度 高度
...

测试样例举例：

1.10G

```
8 10
3 1
1 2
6 7
7 4
8 2
5 3
6 3
5 1
2 4
8 4
```

2.20W

```
13 20
9 6
6 4
11 3
7 3
13 7
13 5
6 4
5 2
12 5
9 1
12 1
12 7
13 2
11 4
13 8
9 9
2 2
11 2
3 7
8 9
```

2.源代码

为了减少报告长度，源码部分已移除，请移步code文件夹中

Declaration

We hereby declare that all the work done in this project titled "Texture Packing" is of our independent effort.