****

**Kauno technologijos universitetas**

Informatikos fakultetas

**Pirmo laboratorinio darbo ataskaita**

1LD individuali užduotis

|  |  |
| --- | --- |
|  |  |
| **Aistis Jakutonis**  Studentas | (parašas) (data) |
|  |  |
| **Dalius Makackas**  **Andrius Kriščiūnas**  **Tadas Kraujalis**  **Vidmantas Rimavičius**  Dėstytojai | (parašas) (data) |
|  |  |

**Kaunas, 2025**

Turinys

[1. Skirtingos paveikslėlio generavimo realizacijos 3](#_Toc192093100)

[1.1. Realizacija, kai paveikslėlio matmenys yra nekeičiami, o nurodomas skirtingas rekursijos gylis 3](#_Toc192093101)

[1.2. Realizacija, kai paveikslėlio matmenys kinta, o rekursijos gylis lieka pastovus 6](#_Toc192093102)

[2. Eksperimentiškai nustatyta priklausomybė pavaizduota grafikais 9](#_Toc192093103)

[2.1. Laiko priklausomybė nuo rekursijos gylio 9](#_Toc192093104)

[2.2. Laiko priklausomybė nuo paveikslėlio matmenų 10](#_Toc192093105)

[2.3. Veiksmų skaičiaus priklausomybė nuo rekursijos gylio 10](#_Toc192093106)

[2.4. Veiksmų skaičiaus priklausomybė nuo paveikslėlio matmenų 11](#_Toc192093107)

[3. Analitinis įvertinimas procedūrų, kurios generuoja paveikslėlius, pagal veiksmų skaičių ir laiką 13](#_Toc192093108)

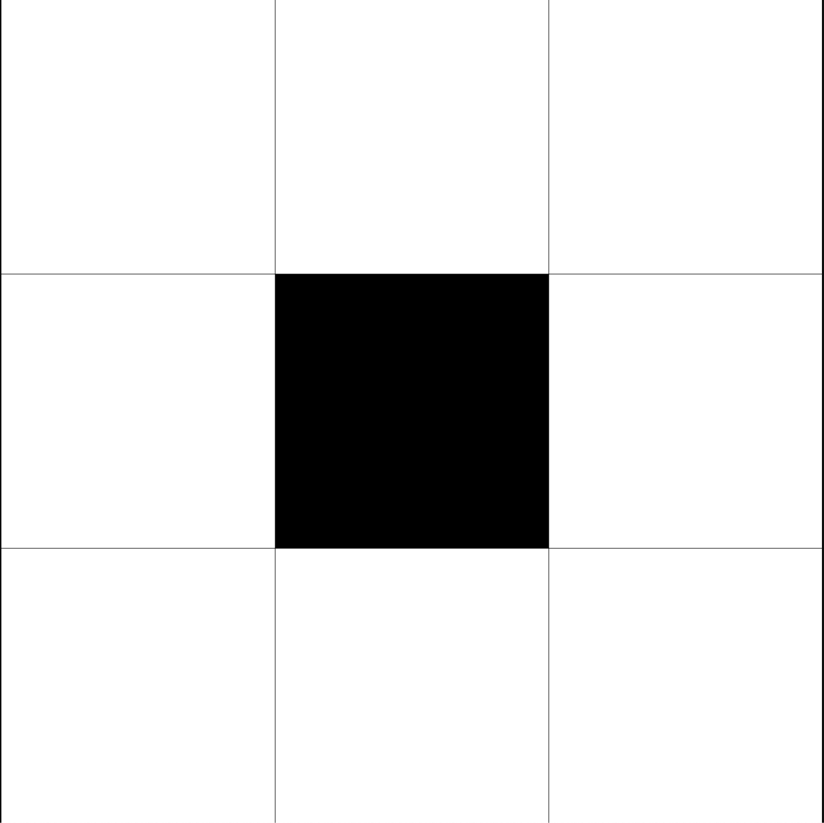
[3.1. Sudaromos ir išsprendžiamos rekurentinės lygtys 13](#_Toc192093109)

[3.2. Rezultatų palyginimas su eksperimentiškai gautais rezultatais 17](#_Toc192093110)

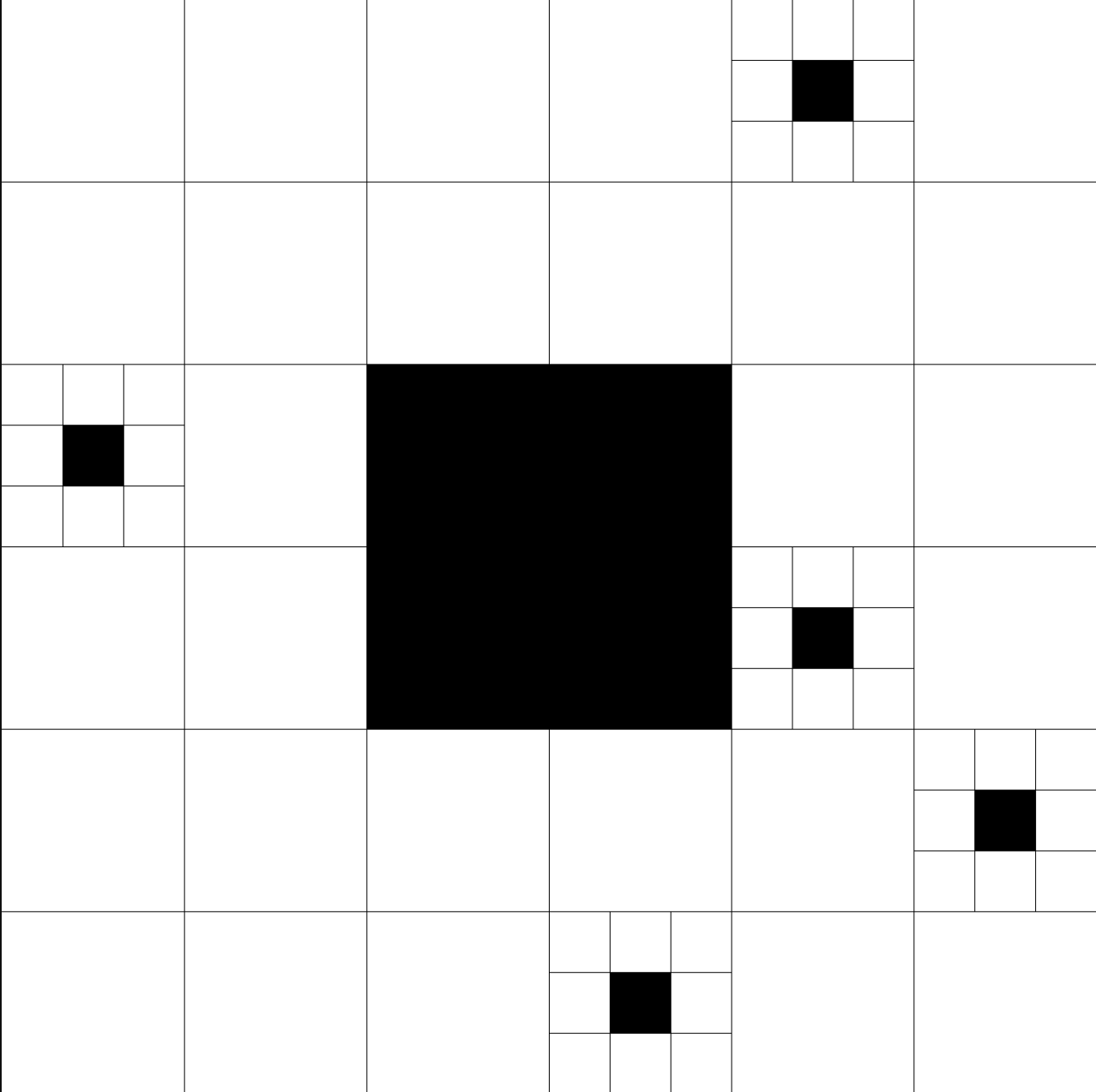
# Skirtingos paveikslėlio generavimo realizacijos

## Realizacija, kai paveikslėlio matmenys yra nekeičiami, o nurodomas skirtingas rekursijos gylis

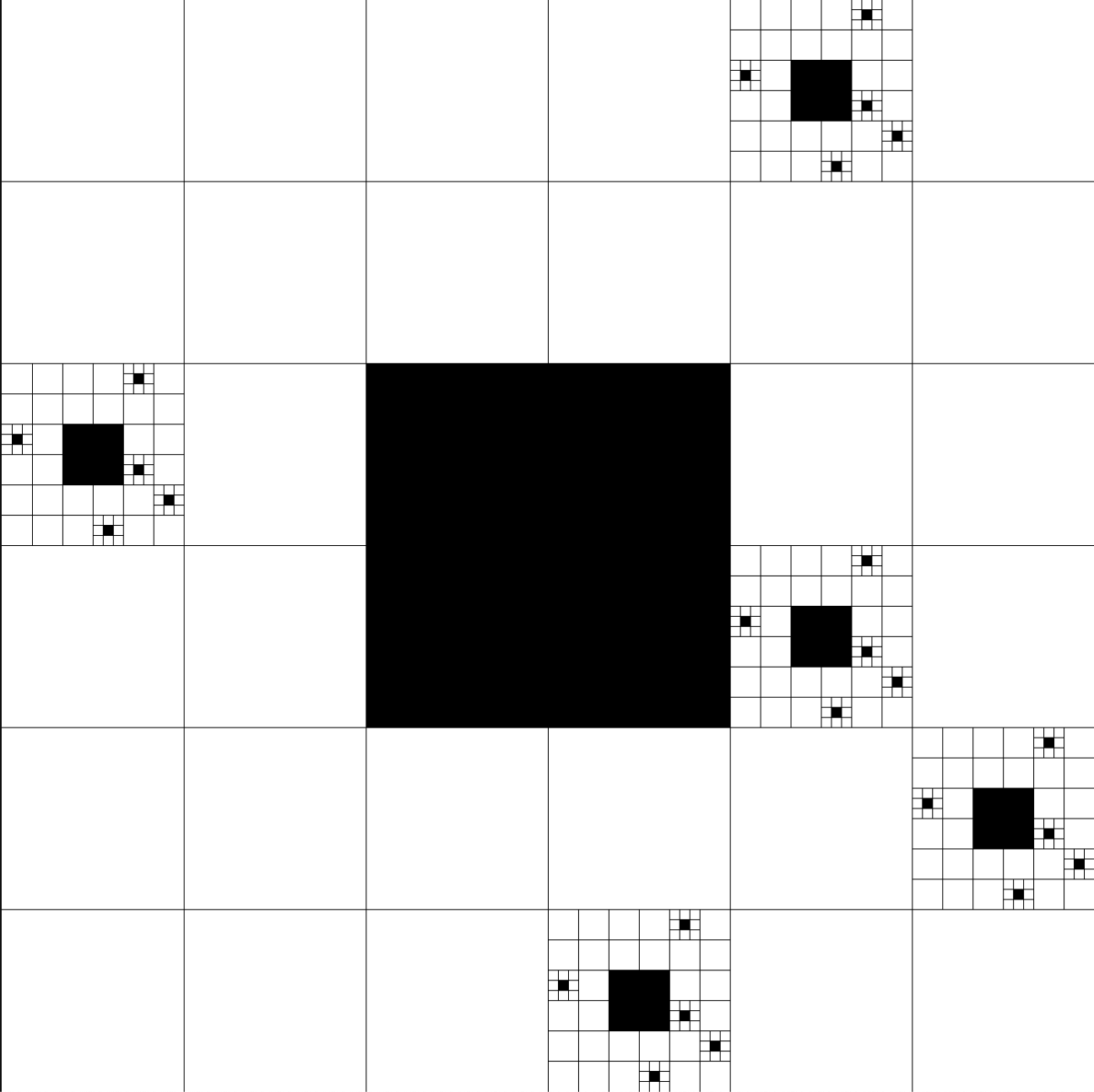
Šiam variantui paveikslėlio matmenys buvo pasirinkti tokie: 1296x1296. O rekursijos gyliai buvo pasirinkti šie: 0; 1; 2; 3; 4; 12.



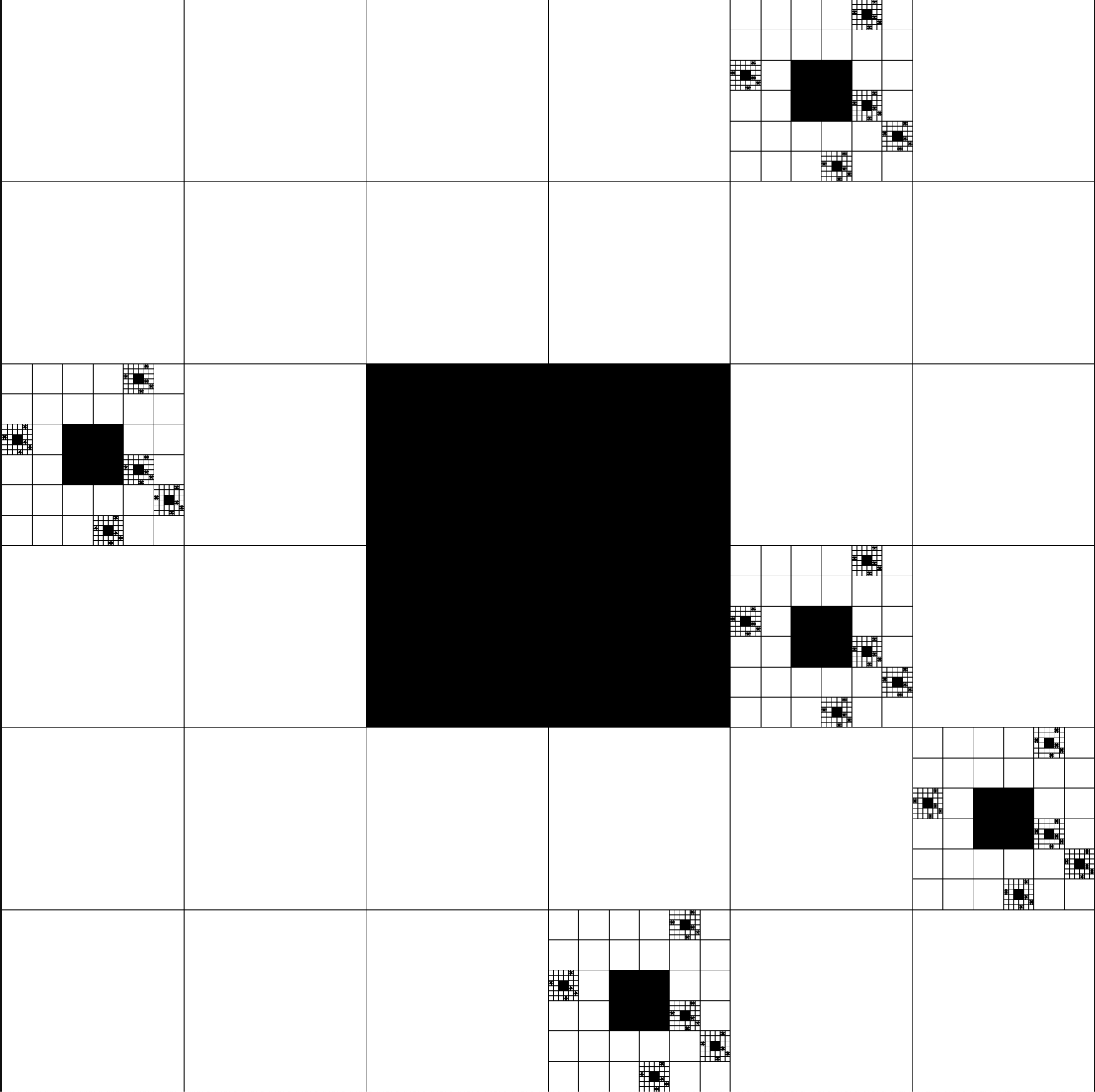
1 pav. Rekursijos gylis lygus 0



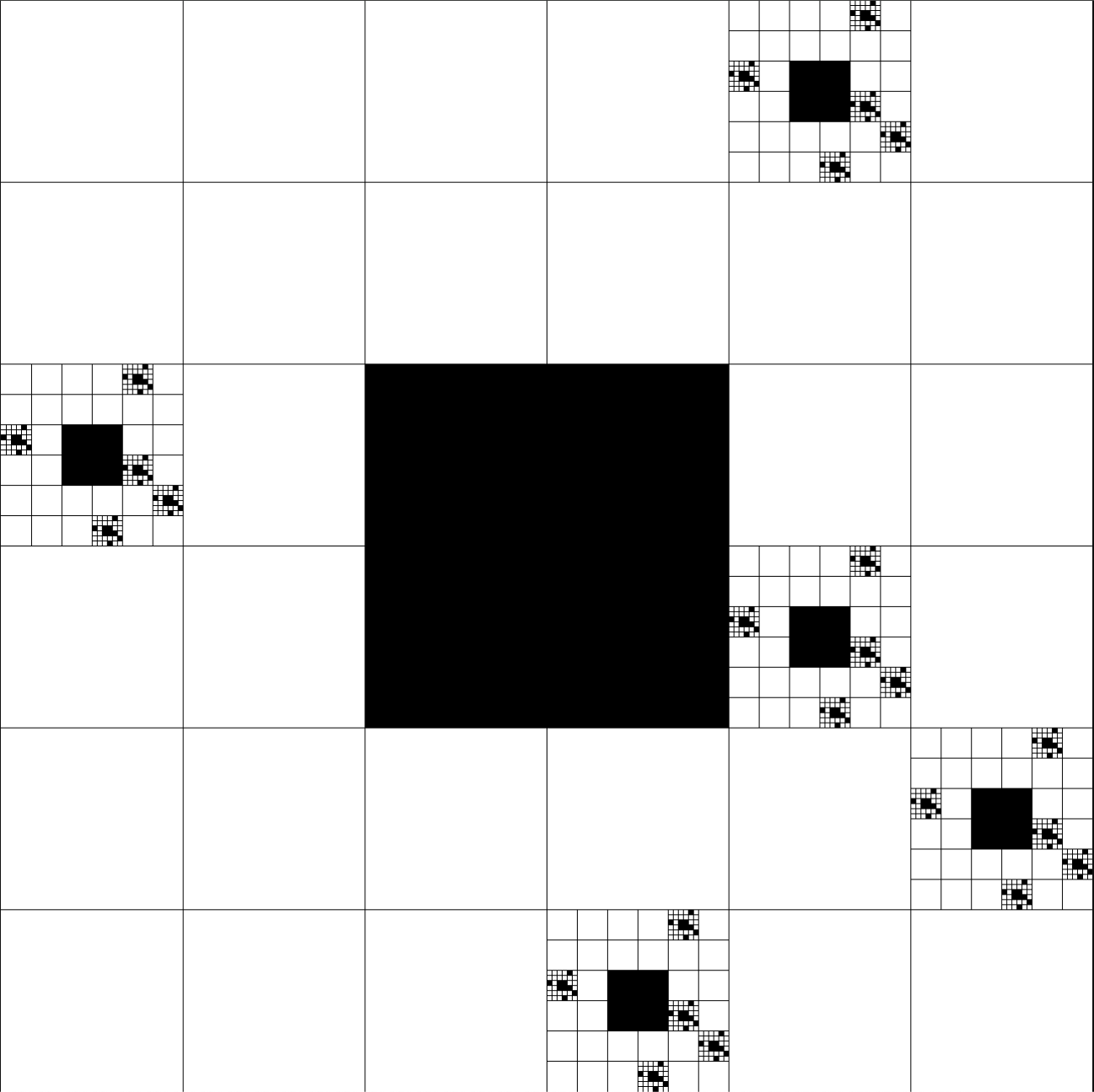
2 pav. Rekursijos gylis lygus 1



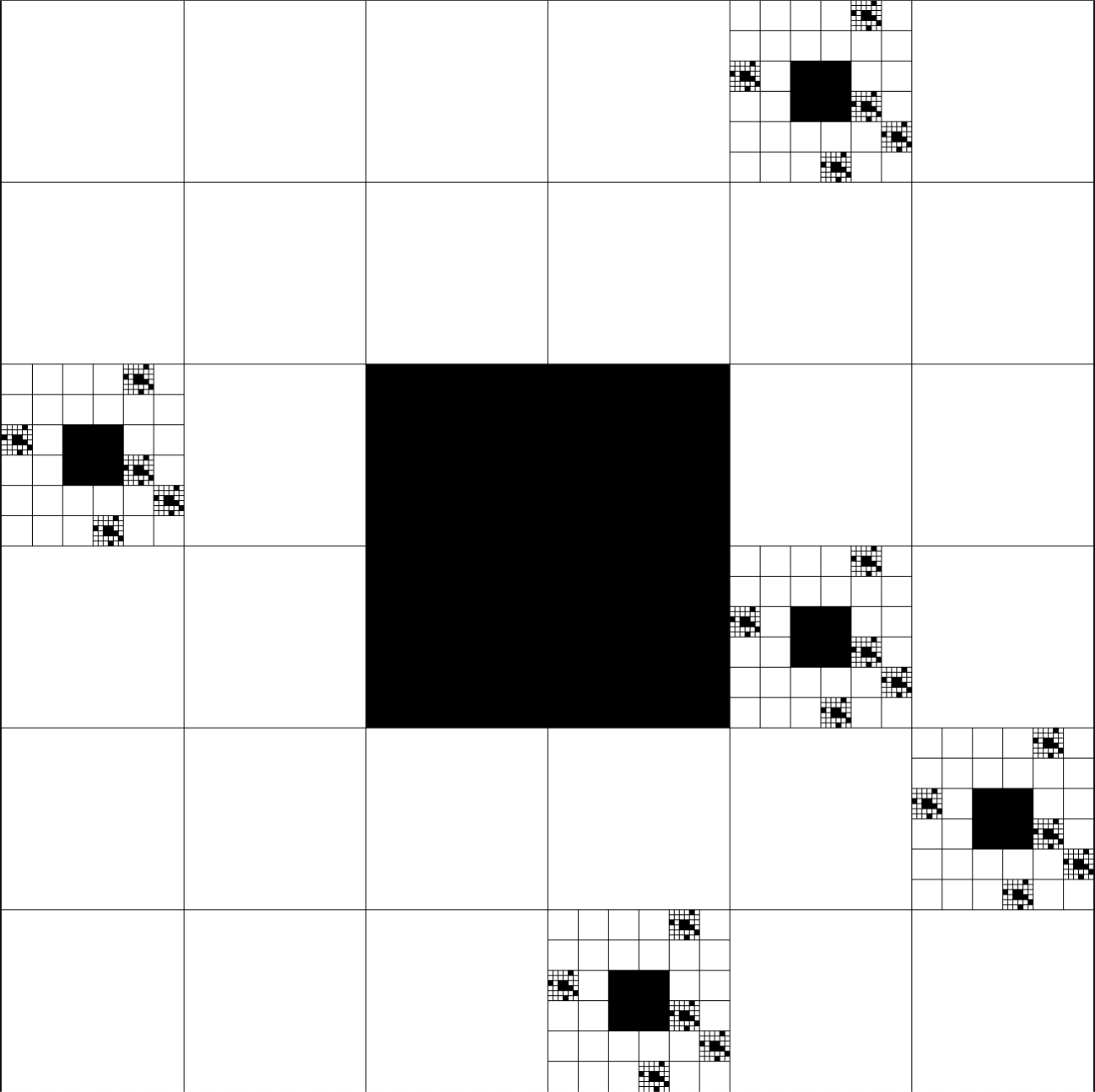
3 pav. Rekursijos gylis lygus 2



4 pav. Rekursijos gylis lygus 3



5 pav. Rekursijos gylis lygus 4

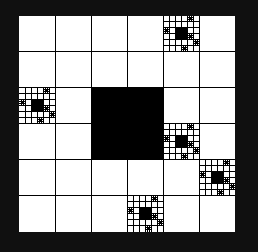


6 pav. Rekursijos gylis lygus 12

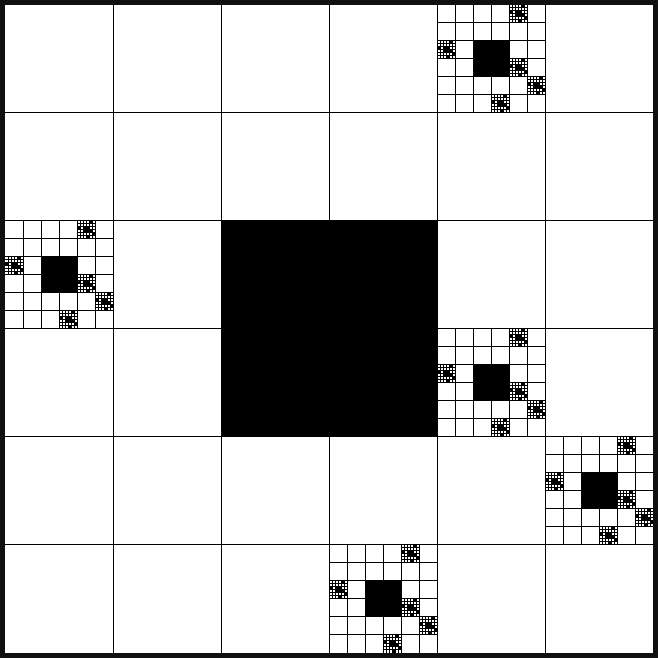
## Realizacija, kai paveikslėlio matmenys kinta, o rekursijos gylis lieka pastovus

Čia bus vykdoma rekursija tol kol nebebus įmanoma jos vykdyti.

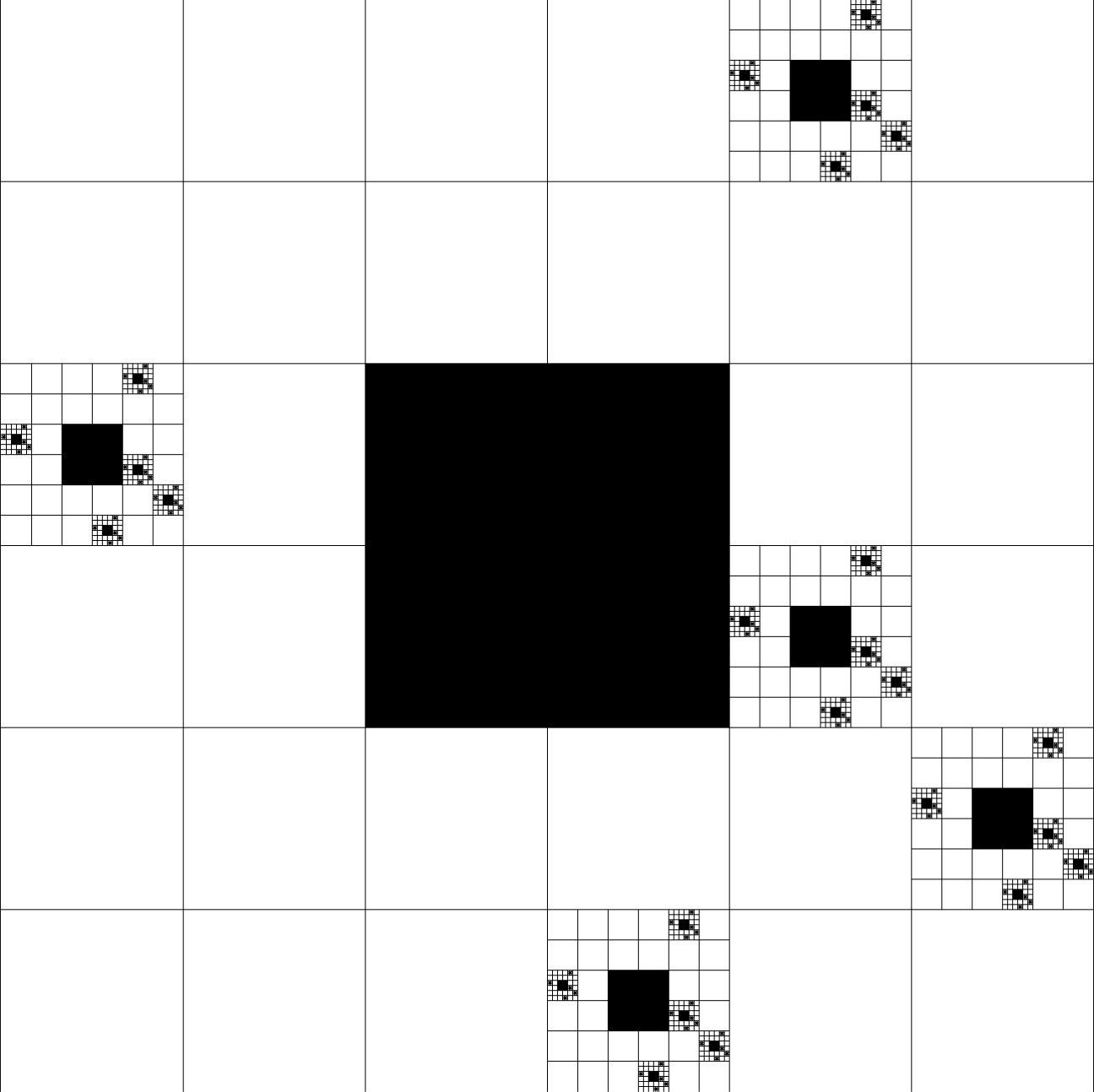
Šiuo atveju buvo nustatyta, kad rekursijos gylis bus 15. O paveikslėlio matmenys kis taip: 216x216; 648x648; 1296x1296; 7776x7776.



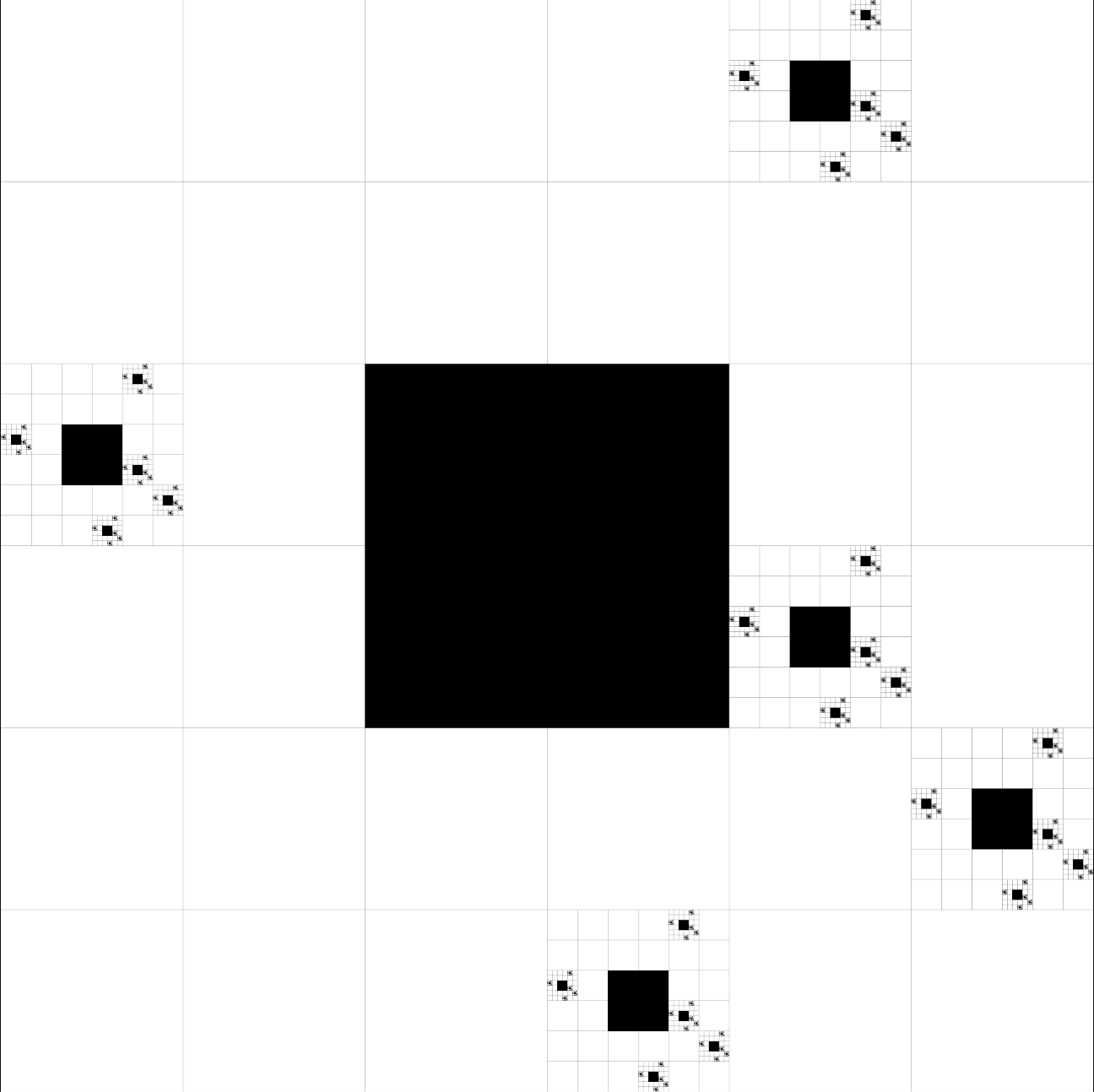
7 pav. Paveikslėlio matmenys lygūs 216x216



8 pav. Paveikslėlio matmenys lygūs 648x648



9 pav. Paveikslėlio matmenys lygūs 1296x1296



10 pav. Paveikslėlio matmenys lygūs 7776x7776

# Eksperimentiškai nustatyta priklausomybė pavaizduota grafikais

## Laiko priklausomybė nuo rekursijos gylio

Paveikslėlio matmenys išlieka tokie patys (1296x1296). Kintant rekursijos gyliui kinta ir laikas. Šią priklausomybę atvaizduojame grafiku.

1 lentelė Laiko priklausomybės nuo rekursijos gylio grafiko duomenys

|  |  |
| --- | --- |
| Recursion depth, number | Time, ms |
| 0 | 14 |
| 1 | 15 |
| 2 | 15 |
| 4 | 16 |
| 8 | 25 |
| 12 | 6864 |

1 grafikas Laiko priklausomybė nuo rekursijos gylio

Iš duomenų matosi, kad didėjant rekursijos gyliui laikas taip pat didėja.

## Laiko priklausomybė nuo paveikslėlio matmenų

Paveikslėlio matmenys kinta, o paveikslėlis yra rekursyviai piešiamas iki tol kol nebėra, kur piešti.

2 lentelė Laiko priklausomybės nuo paveikslėlio matmenų grafiko duomenys

|  |  |
| --- | --- |
| Image resolution, pixels | Time, ms |
| 36 | 3 |
| 216 | 3 |
| 648 | 5 |
| 1296 | 14 |
| 3888 | 108 |
| 7776 | 421 |
| 23328 | 3738 |

2 grafikas Laiko priklausomybė nuo paveikslėlio matmenų

Iš grafiko galime matyti, kad didėjant paveikslėlio matmenims didėja ir laiko sąnaudos.

## Veiksmų skaičiaus priklausomybė nuo rekursijos gylio

Paveikslėlio matmenys nekinta (1296x1296), o rekursijos gylis yra keičiamas, tuo metu keičiasi ir atliekamas veiksmų skaičius.

3 lentelė Veiksmų skaičiaus priklausomybės nuo rekursijos gylio grafiko duomenys

|  |  |  |
| --- | --- | --- |
| Recursion depth, number | Action count, number | Action count in thousands |
| 0 | 10272565 | 10272 |
| 1 | 10311041 | 10311 |
| 2 | 10325521 | 10325 |
| 4 | 10356671 | 10356 |
| 8 | 20110421 | 20110 |
| 12 | 1818893125 | 1818893 |

3 grafikas Veiksmų skaičiaus priklausomybė nuo rekursijos gylio

Grafika parodo, kad didėjant rekursijos gyliui didėja ir veiksmų skaičius.

## Veiksmų skaičiaus priklausomybė nuo paveikslėlio matmenų

Paveikslėlio rekursijos gylis lieka pastovus (25), o jo matmenys kinta. Stebėsime kaip pasikeis veiksmų skaičius kintant paveikslėlio matmenims.

4 lentelė Veiksmų skaičiaus priklausomybės nuo paveikslėlio matmenų grafiko duomenys

|  |  |  |
| --- | --- | --- |
| Image resolution, pixels | Action count, number | Action count in thousands |
| 36 | 8651 | 8 |
| 216 | 291751 | 291 |
| 648 | 2596569 | 2596 |
| 1296 | 10337171 | 10337 |
| 3888 | 92771279 | 92771 |
| 7776 | 370837541 | 370837 |
| 23328 | 3336083789 | 3336083 |

4 grafikas Veiksmų skaičiaus priklausomybė nuo paveikslėlio matmenų

# Analitinis įvertinimas procedūrų, kurios generuoja paveikslėlius, pagal veiksmų skaičių ir laiką

## Sudaromos ir išsprendžiamos rekurentinės lygtys

|  |  |  |  |
| --- | --- | --- | --- |
| Nr. | Kodas | Laikas | Kartai |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58 | **static** **int** widthLimit = **6**;  **static** **void** **Main**(**string**[] args)  {  **int** resolution = **1296**;  **int** recDepth = **3**;  **byte**[,] table = **new** **byte**[resolution,  resolution];  fillWhite(table);  drawRecursively(table, **0**, **0**,  recDepth, resolution);  **byte**[] bpm = createBmp(table);  **var** header = **new** **byte**[**54**]  {  **0**x42, **0**x4d,  **0**x0, **0**x0, **0**x0, **0**x0,  **0**x0, **0**x0, **0**x0, **0**x0,  **0**x0, **0**x0, **0**x0, **0**x0,  **0**x28, **0**x0, **0**x0, **0**x0,  **0**xe8, **0**x3, **0**x0, **0**x0,  **0**xe8, **0**x3, **0**x0, **0**x0,  **0**x1, **0**x0,  **0**x10, **0**x0,  **0**x0, **0**x0, **0**x0, **0**x0,  **0**x0, **0**x0, **0**x0, **0**x0,  **0**x0, **0**x0, **0**x0, **0**x0,  **0**x0, **0**x0, **0**x0, **0**x0,  **0**x0, **0**x0, **0**x0, **0**x0,  **0**x0, **0**x0, **0**x0, **0**x0  };  **using** (FileStream file =  **new** **FileStream**("test.bmp",  FileMode.Create,  FileAccess.Write))  {  Array.Copy(  BitConverter.GetBytes(  (**int**)resolution), **0**,  header, **0**x12,  **sizeof**(**int**));  Array.Copy(  BitConverter.GetBytes(  (**int**)resolution), **0**,  header, **0**x16,  **sizeof**(**int**));  file.Write(header);  file.Write(bpm);  file.Close();  }  } | c1  c2  c2  c3  c4  c5  c6  c6  c7  c8  c9 | 1  1  1  1  1  1  1  1  1 |
|  | | | |
| 59  60  61  62  63  64 | **static** **void** **fillWhite**(**byte**[,] table)  {  drawColorSquare(**0**, **0**,  table.GetLength(**0**),  table, **1**);  } |  |  |
|  | | | |
| 65  66  67  68  69  70  71  72  73  74  75 | **static** **void** **drawColorSquare**(**int** x, **int** y,  **int** width, **byte**[,] table, **byte** color)  {  **for** (**int** i = y; i < y + width; i++)  {  **for** (**int** j = x; j < x + width; j++)  {  table[i, j] = color;  }  }  } | c1  c2  c3 |  |
|  | | | |
| 76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105 | **static** **void** **drawRecursively**(**byte**[,] table,  **int** x, **int** y, **int** recDepth, **int** width)  {  **if** (recDepth == **0**)  {  drawFinish(x, y, width, table);  **return**;  }  **if** (width <= widthLimit)  {  drawFinish(x, y, width, table);  **return**;  }  drawPrimary(x, y, width, table);  wid = width / **6**;  drawRecursively(table, x + wid \* **4**,  y + wid \* **5**, recDepth - **1**, wid);  drawRecursively(table, x, y + wid \* **3**,  recDepth - **1**, wid);  drawRecursively(table, x + wid \* **4**,  y + wid \* **2**, recDepth - **1**, wid);  drawRecursively(table, x + wid \* **5**,  y + wid, recDepth - **1**, wid);  drawRecursively(table, x + wid \* **3**,  y, recDepth - **1**, wid);  } | c1  c3  c4  c3  c5 | 1,  1, X1  1,  1, X2  – X1X2  1 – X1X2  arba  arba  arba  arba  arba |
|  | | | |
| 106  107  108  109  110  111  112 | **static** **void** **drawFinish**(**int** x, **int** y,  **int** width, **byte**[,] table)  {  **int** w3 = width / **3**;  drawSquare(x + w3, y + w3, w3, table);  drawFinishGrid(x, y, width, table);  } | c1 | 1 |
|  | | | |
| 113  114  115  116  117  118  119  120  121  122  123 | **static** **void** **drawFinishGrid**(**int** x, **int** y,  **int** width, **byte**[,] table)  {  **for** (**int** i = **1**; i < **3**; i++)  {  drawVertical(x + width / **3** \* i,  y, width, table);  drawHorizontal(x,  y + width / **3** \* i, width, table);  }  } | c1 | 3 |
|  | | | |
| 124  125  126  127  128  129  130 | **static** **void** **drawPrimary**(**int** x, **int** y,  **int** width, **byte**[,] table)  {  **int** w3 = width / **3**;  drawSquare(x + w3, y + w3, w3, table);  drawGrid(x, y, width, table);  } | c1 | 1 |
|  | | | |
| 131  132  133  134  135 | **static** **void** **drawSquare**(**int** x, **int** y,  **int** width, **byte**[,] table)  {  drawColorSquare(x, y, width, table, **0**);  } |  |  |
|  | | | |
| 136  137  138  139  140  141  142  143  144  145  146  147 | **static** **void** **drawGrid**(**int** x, **int** y,  **int** width, **byte**[,] table)  {  **int** w6 = width / **6**;  **for** (**int** i = **1**; i < **6**; i++)  {  drawVertical(x + w6 \* i, y,  width, table);  drawHorizontal(x, y + w6 \* i,  width, table);  }  } | c1  c2 | 1  6 |
|  | | | |
| 148  149  150  151  152  153  154  155 | **static** **void** **drawVertical**(**int** x, **int** y,  **int** width, **byte**[,] table)  {  **for** (**int** i = y; i < y + width; i++)  {  table[i, x] = **0**;  }  } | c1  c2 |  |
|  | | | |
| 156  157  158  159  160  161  162  163 | **static** **void** **drawHorizontal**(**int** x, **int** y,  **int** width, **byte**[,] table)  {  **for** (**int** i = x; i < x + width; i++)  {  table[y, i] = **0**;  }  } | c1  c2 |  |
|  | | | |
| 164  165  166  167  168  169  170  171  172  173  174  175  176  177  178  179  180  181  182  183  184  185  186  187 | **static** **byte**[] **createBmp**(**byte**[,] table)  {  **int** width = table.GetLength(**0**);  **long** line = (width \* **16** + **31**) / **32** \* **4**;  **byte**[] bpm = **new** **byte**[width \* line];  **for** (**int** y = **0**; y < width; y++)  {  **for** (**int** x = **0**; x < width; x++)  {  **ushort** pixel = table[y, x];  **ushort** color = (pixel == **1**) ?  (**ushort**)**0**xFFFF :  (**ushort**)**0**x0000;  **long** pos = y \* line + x \* **2**;  bpm[pos] = (**byte**)  (color & **0**xFF);  bpm[pos + **1**] = (**byte**)  ((color >> **8**) & **0**xFF);  }  }  **return** bpm;  } | c1  c2  c3  c4  c5  c6  c7  c8  c9  c10  c11 | 1  1  1  1 |
|  | | | |

## Rezultatų palyginimas su eksperimentiškai gautais rezultatais