Лекция 12

- Вычислительные шейдеры (продолжение).
- Open**GL** **S**hading **L**anguage.

Вычислительные шейдеры — в обход OpenGL конвейера

util_template.cpp

```
void   csDataInit(GLuint* , int );

int initBuffer(){

  glGenBuffers( 2,bufferID);
  glBindBuffer( GL_ARRAY_BUFFER, bufferID[0]);
  glBufferData( GL_ARRAY_BUFFER, 6*num_of_verticies*sizeof(float),
                                                    0, GL_DYNAMIC_DRAW );

  glBindBuffer( GL_ARRAY_BUFFER, bufferID[1]);
  glBufferData( GL_ARRAY_BUFFER, 3*num_of_verticies*sizeof(float),
                                                    0, GL_DYNAMIC_DRAW );



  csDataInit(bufferID, num_of_verticies);



  return 0;
}
```

csh_template.cpp

```
void csDataInit(GLuint* inBuf,int N){

  glBindBufferBase(GL_SHADER_STORAGE_BUFFER, 0, inBuf[0]);
  glBindBufferBase(GL_SHADER_STORAGE_BUFFER, 1, inBuf[1]);

  GLuint computeShaderID=genComputeProg();
  glUseProgram(computeShaderID);

  glDispatchCompute(N/128, 1, 1);

  glMemoryBarrier( GL_SHADER_STORAGE_BARRIER_BIT);
}
```

```
GLuint genComputeProg(){
    GLuint progHandle = glCreateProgram();
    GLuint cs = glCreateShader(GL_COMPUTE_SHADER);

const char *cpSrc[] = {
      "#version 430\n",
      "layout (local_size_x = 128, local_size_y = 1, local_size_z = 1) in; \
       layout(std430, binding = 0) buffer PositionBuffer{float Pos[];};\
       layout(std430, binding = 1) buffer VelocityBuffer{float Vel[];};\
 float lmap(in uint i){\
        uint count;\
        float x=0.78;\
        for(count=0;count<i;count++)\
          x=3.99*x*(1-x);\
        return x;\
       }\
void main() {\
        uint index = gl_GlobalInvocationID.x;\
        Pos[index*6]=-0.5+1.0*lmap(index);\
        Pos[index*6+1]=-0.5+1.0*lmap(index*10);\
        Pos[index*6+2]=0.0;\
        Pos[index*6+3]=1.0;\
        Pos[index*6+4]=1.0;\
        Pos[index*6+5]=0.0;\
        Vel[3*index]=-0.5+1.0*lmap(index);\
        Vel[3*index+1]=-0.5+1.0*lmap(index*10);\
        Vel[3*index+2]=0.0;\
      }"
  };
....................................................................................
```

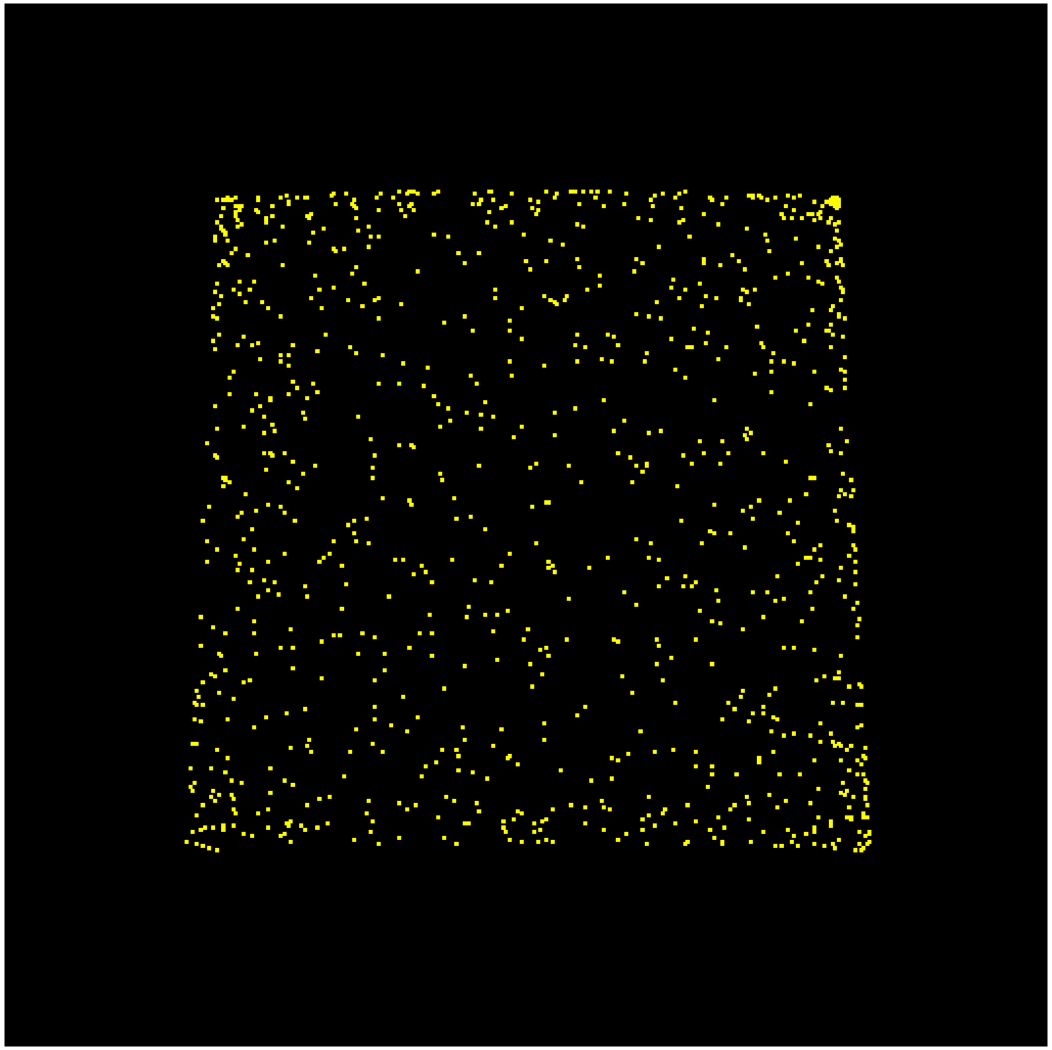
csh_move.cpp

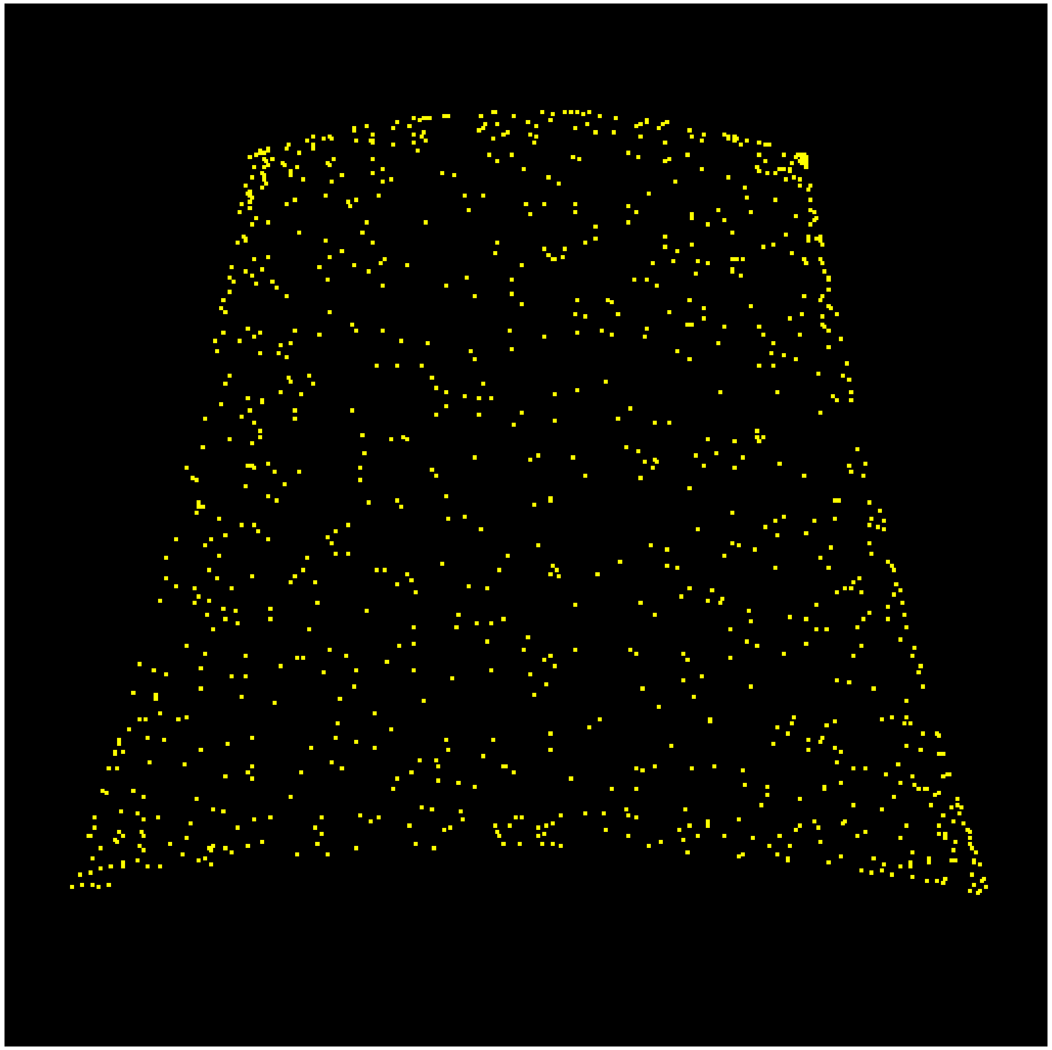
```
……………………………………………………………………………………
GLuint genMoveProg();

void csMove(GLuint* inBuf,int N){

  glBindBufferBase(GL_SHADER_STORAGE_BUFFER, 0, inBuf[0]);
  glBindBufferBase(GL_SHADER_STORAGE_BUFFER, 1, inBuf[1]);

  GLuint computeShaderID=genMoveProg();
  glUseProgram(computeShaderID);

  glDispatchCompute(N/128, 1, 1);

  glMemoryBarrier( GL_SHADER_STORAGE_BARRIER_BIT);
}
```

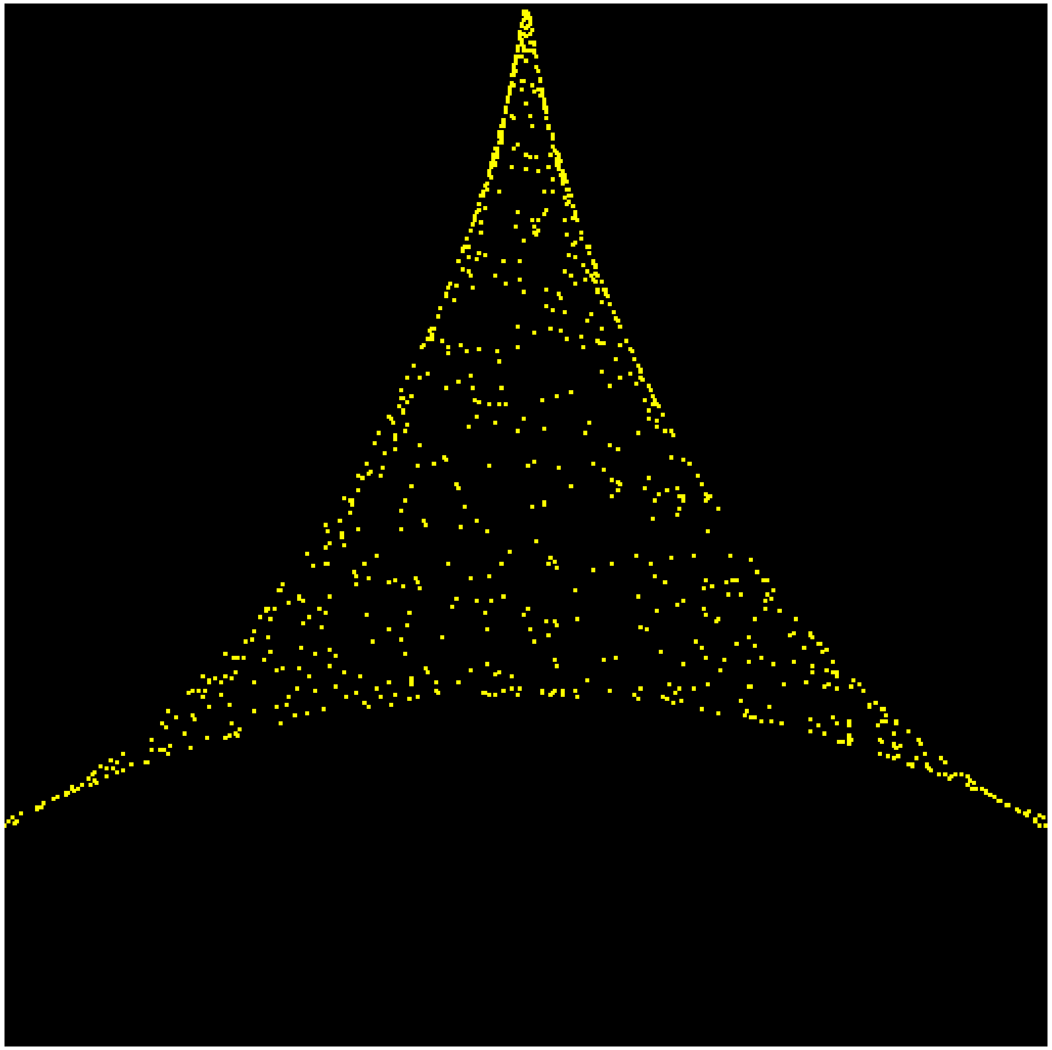
```
GLuint genMoveProg(){
    GLuint progHandle = glCreateProgram();
    GLuint cs = glCreateShader(GL_COMPUTE_SHADER);

    const char *cpSrc[] = {
      "#version 430\n",
      "layout (local_size_x = 128, local_size_y = 1, local_size_z = 1) in; \
       layout(std430, binding = 0) buffer PositionBuffer{float Pos[];};\
       layout(std430, binding = 1) buffer VelocityBuffer{float Vel[];};\
void main() {\
       float x,y,vx,vy;\
       float tau=0.01;\
       float c=2.0;\
       float eps=0.1;\
       uint index = gl_GlobalInvocationID.x;\
       x=Pos[index*6];\
       y=Pos[index*6+1];\
       vx=Vel[3*index];\
       vy=Vel[3*index+1];\
       \
       vx=vx+tau*(-x-eps*(2*x*y));\
       vy=vy+tau*(-y-eps*(x*x-y*y));\
       x=x+tau*vx;\
       y=y+tau*vy;\
       \
       Pos[index*6]=x;\
       Pos[index*6+1]=y;\
       Vel[3*index]=vx;\
       Vel[3*index+1]=vy;\
     }"
};
.........................................................................................
```

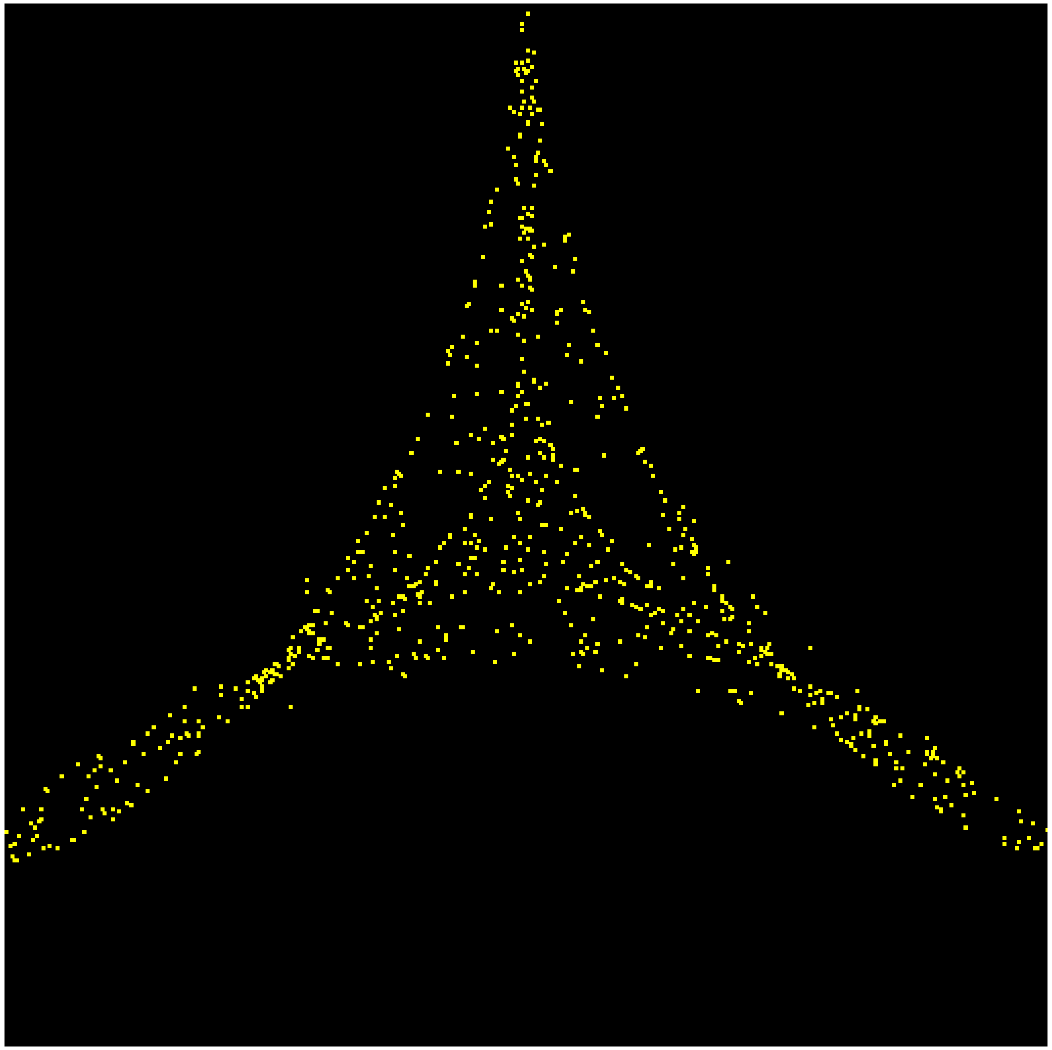
util_template.cpp

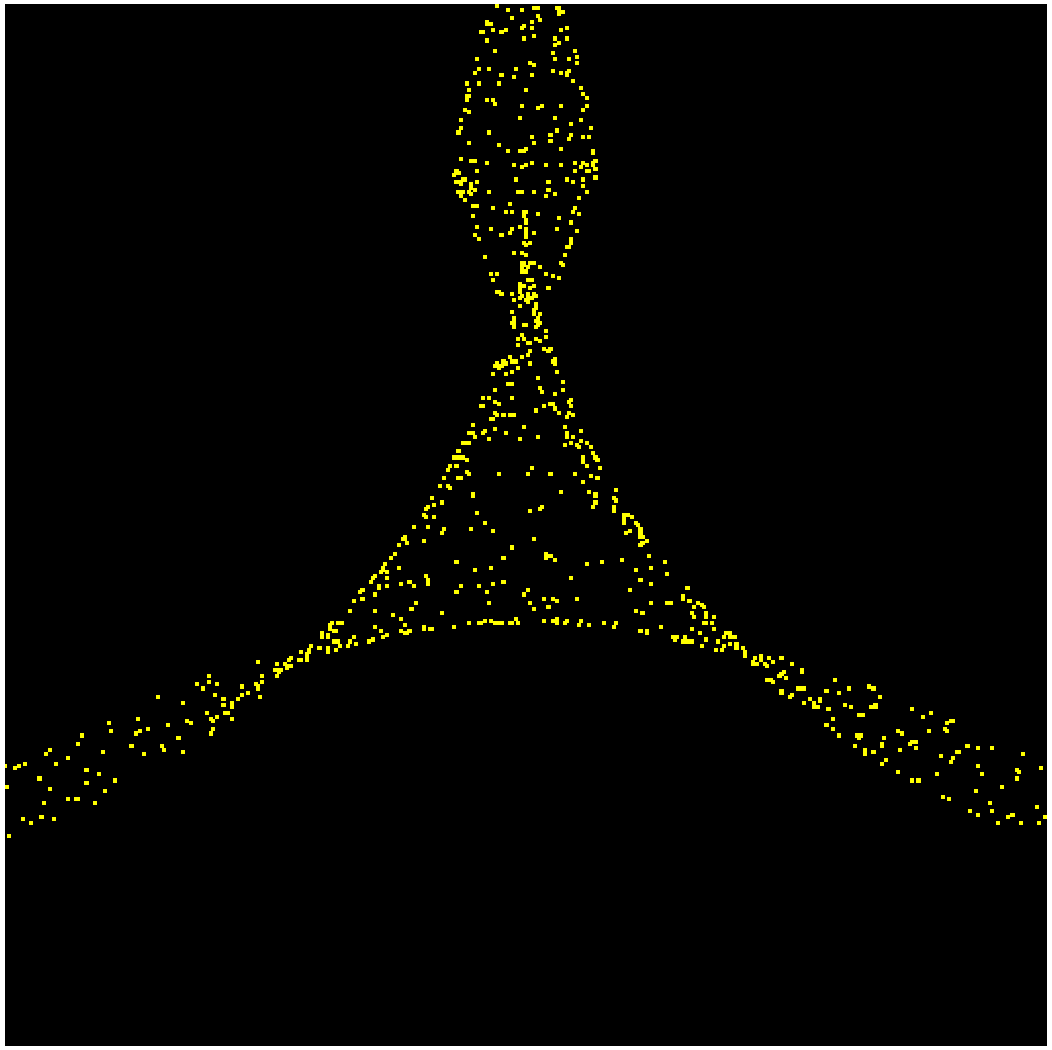
```
void display(){
  glBindBuffer( GL_ARRAY_BUFFER, bufferID[0]);
…………………………………………………………………………………………..
  glDrawArrays(GL_POINTS,0, num_of_verticies);
  glDisableVertexAttribArray(posPtr);
  glDisableVertexAttribArray(colorPtr);

  csMove(bufferID,num_of_verticies);

  void myCleanup(){
     glDeleteBuffers(2, bufferID);
     glDeleteProgram(progHandle);
}
```

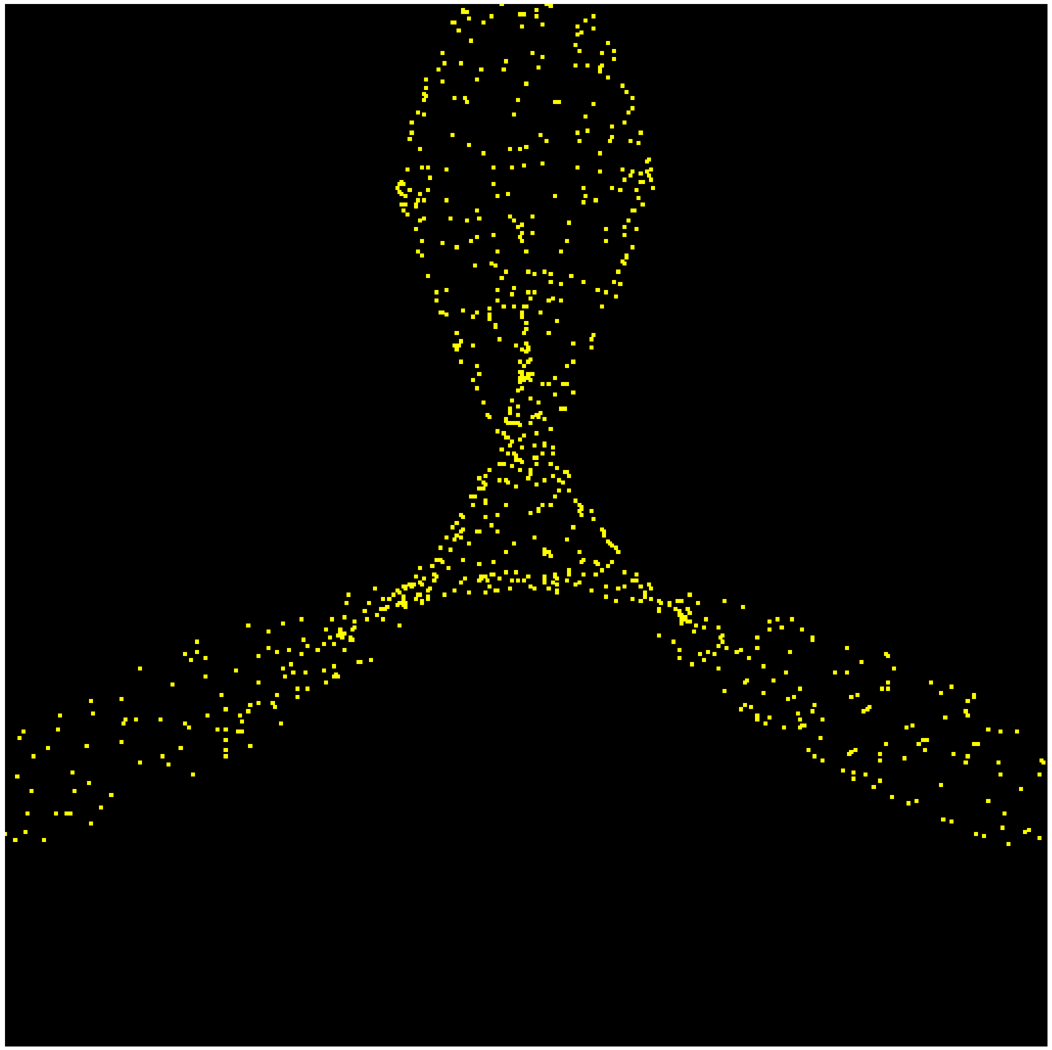


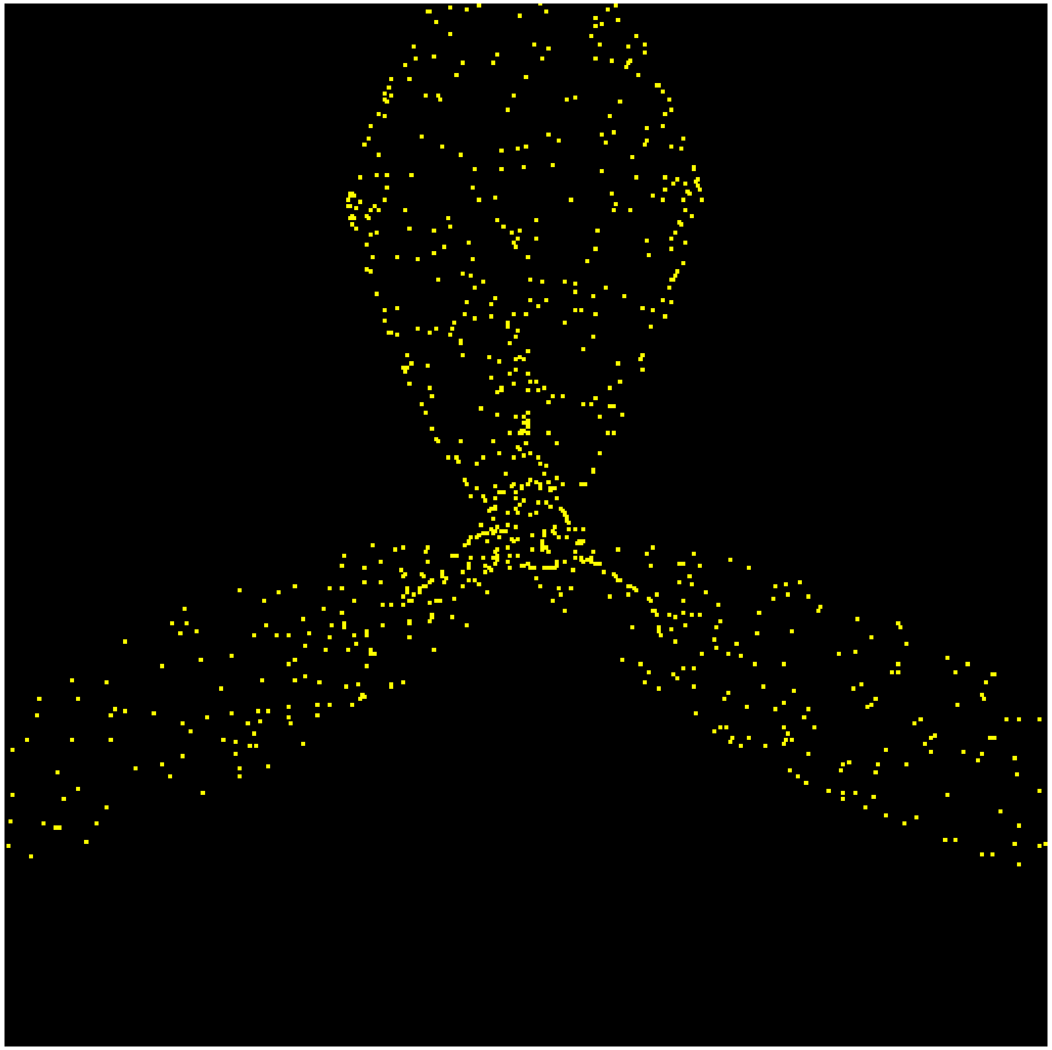


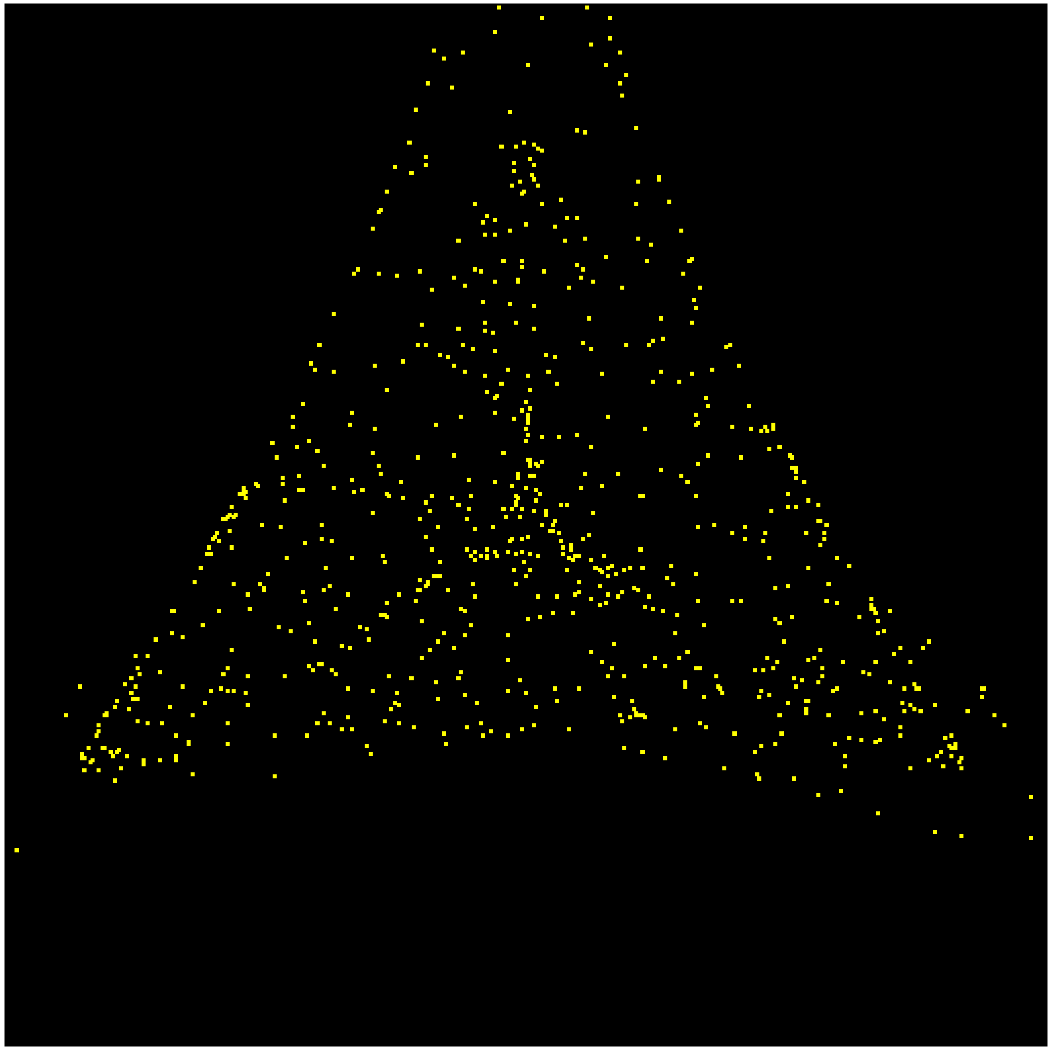


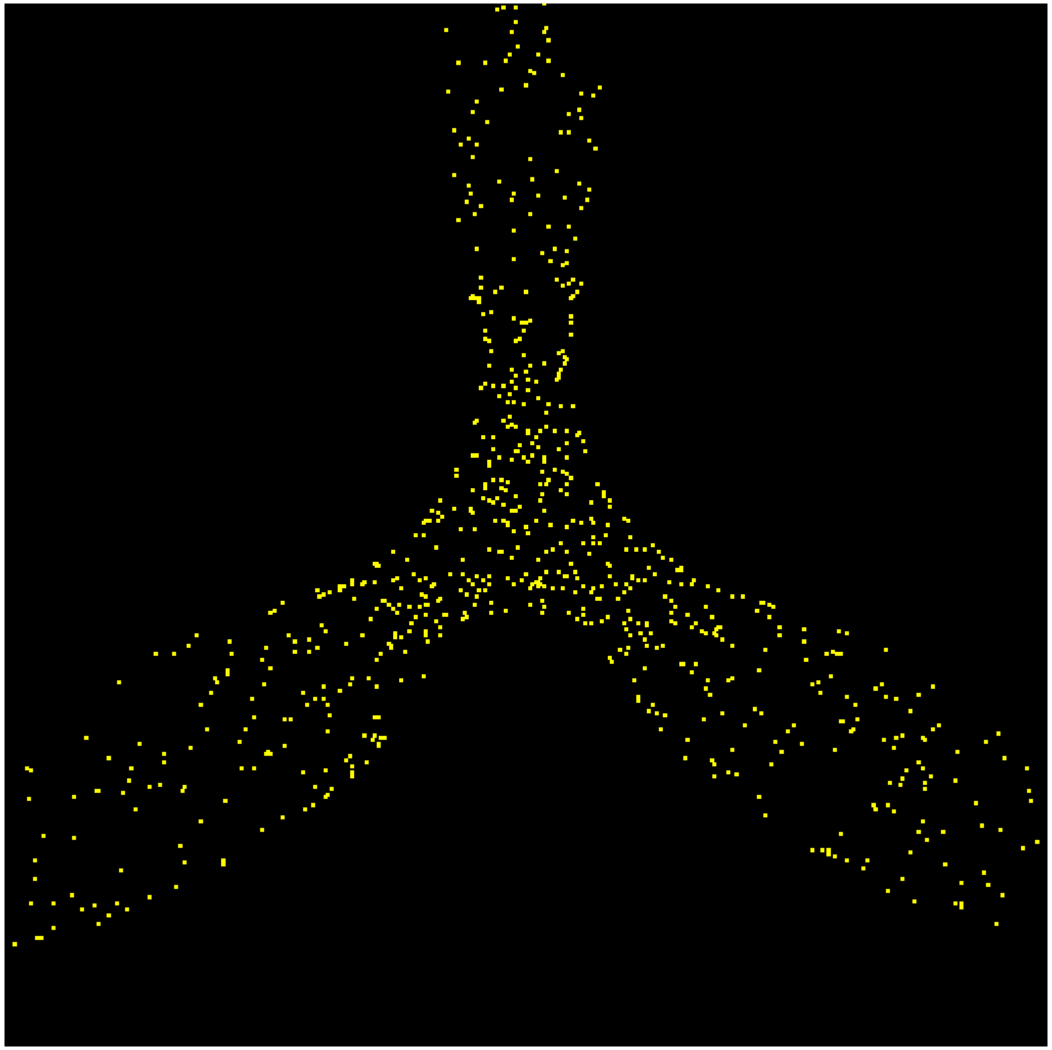


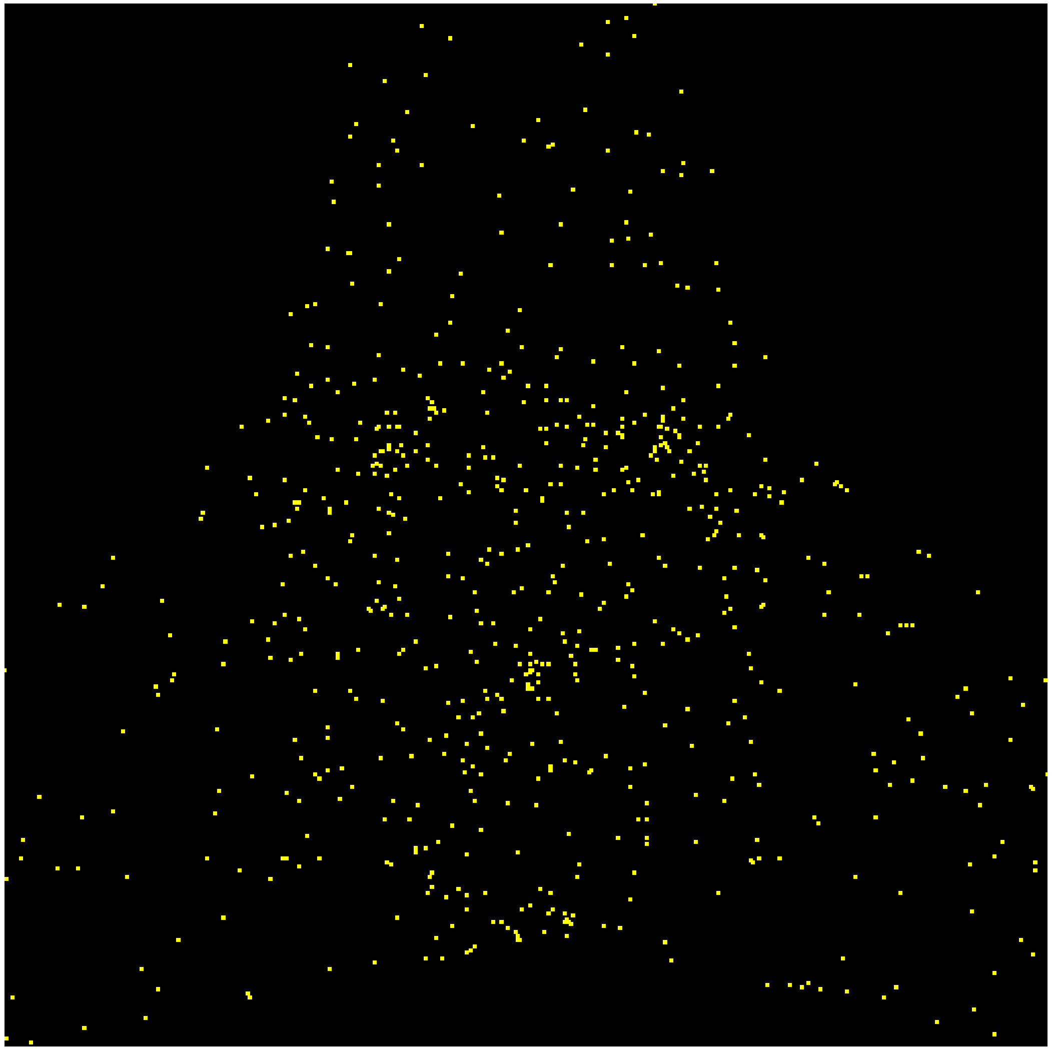


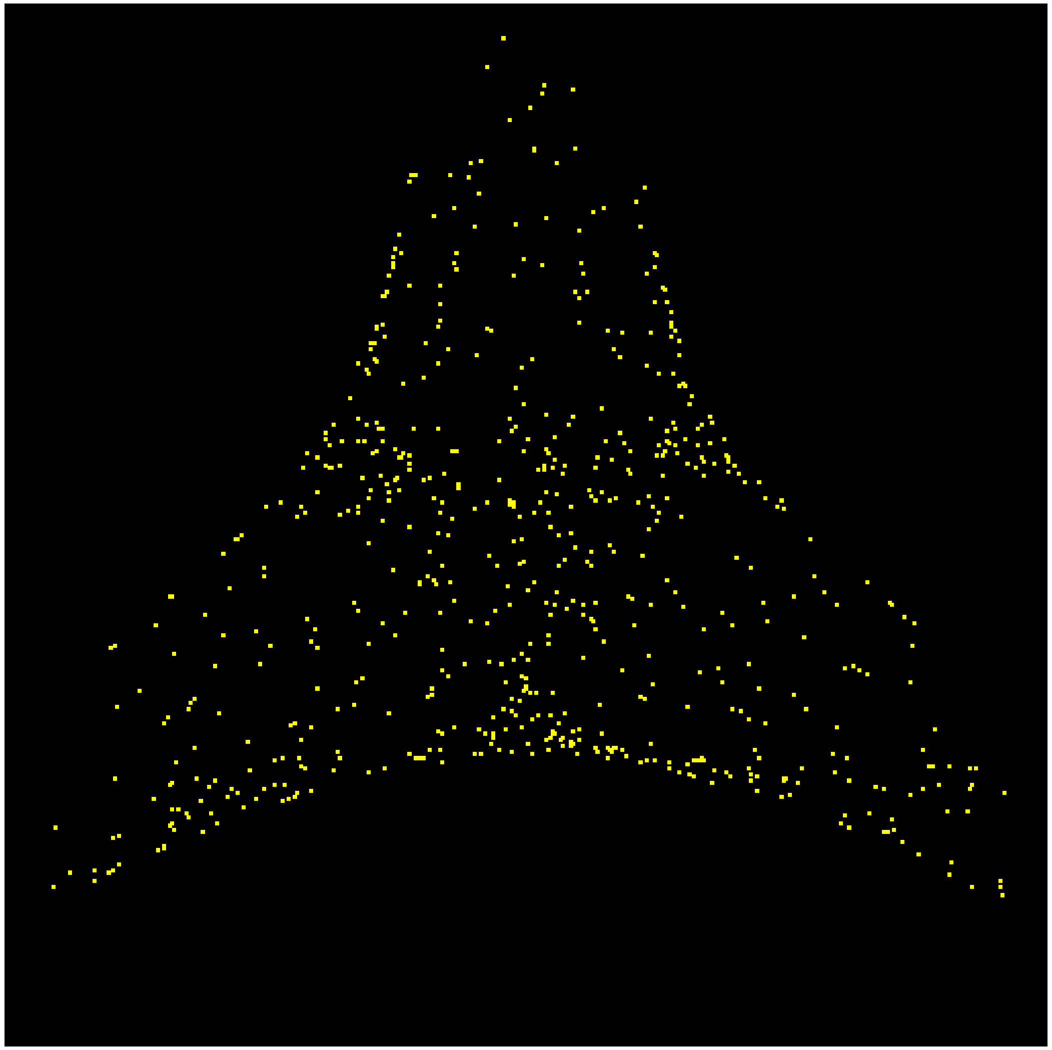


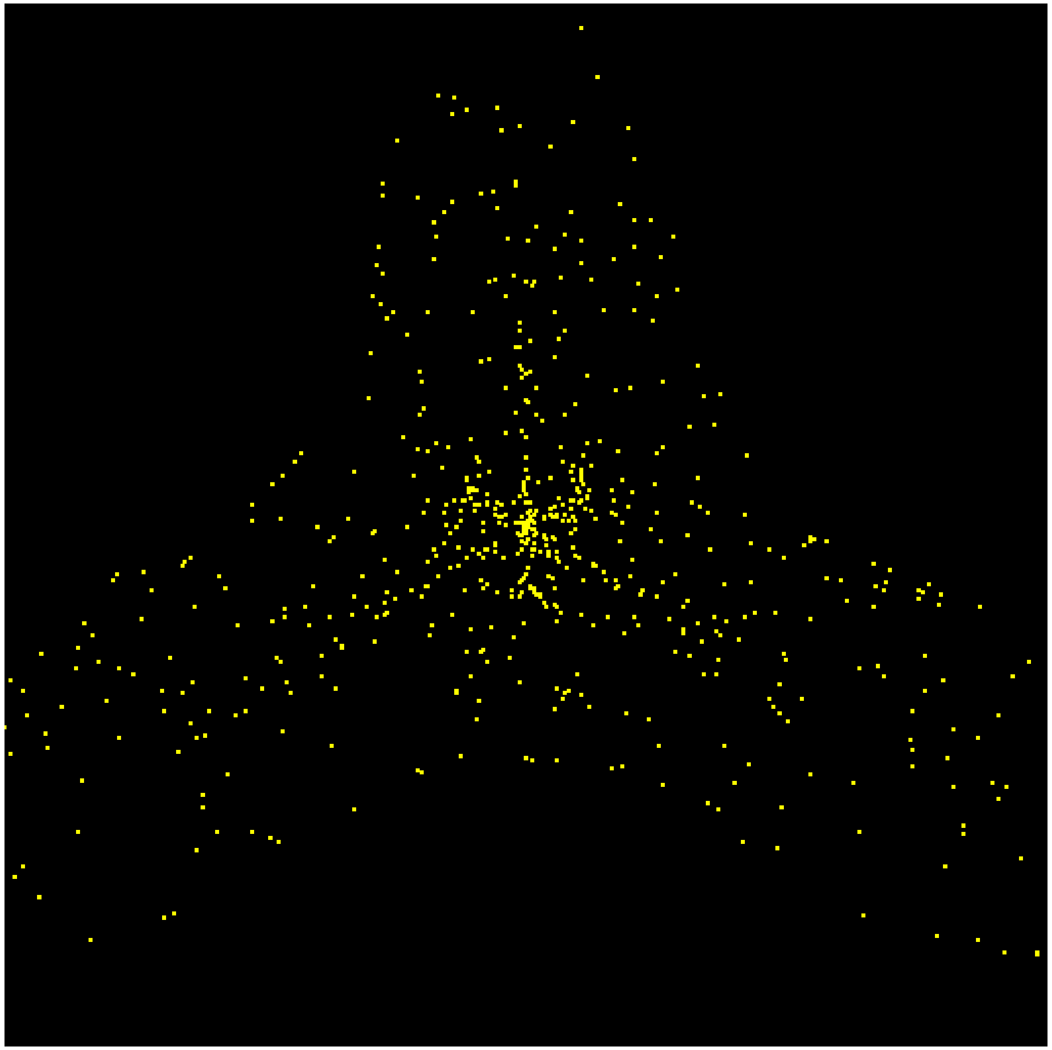


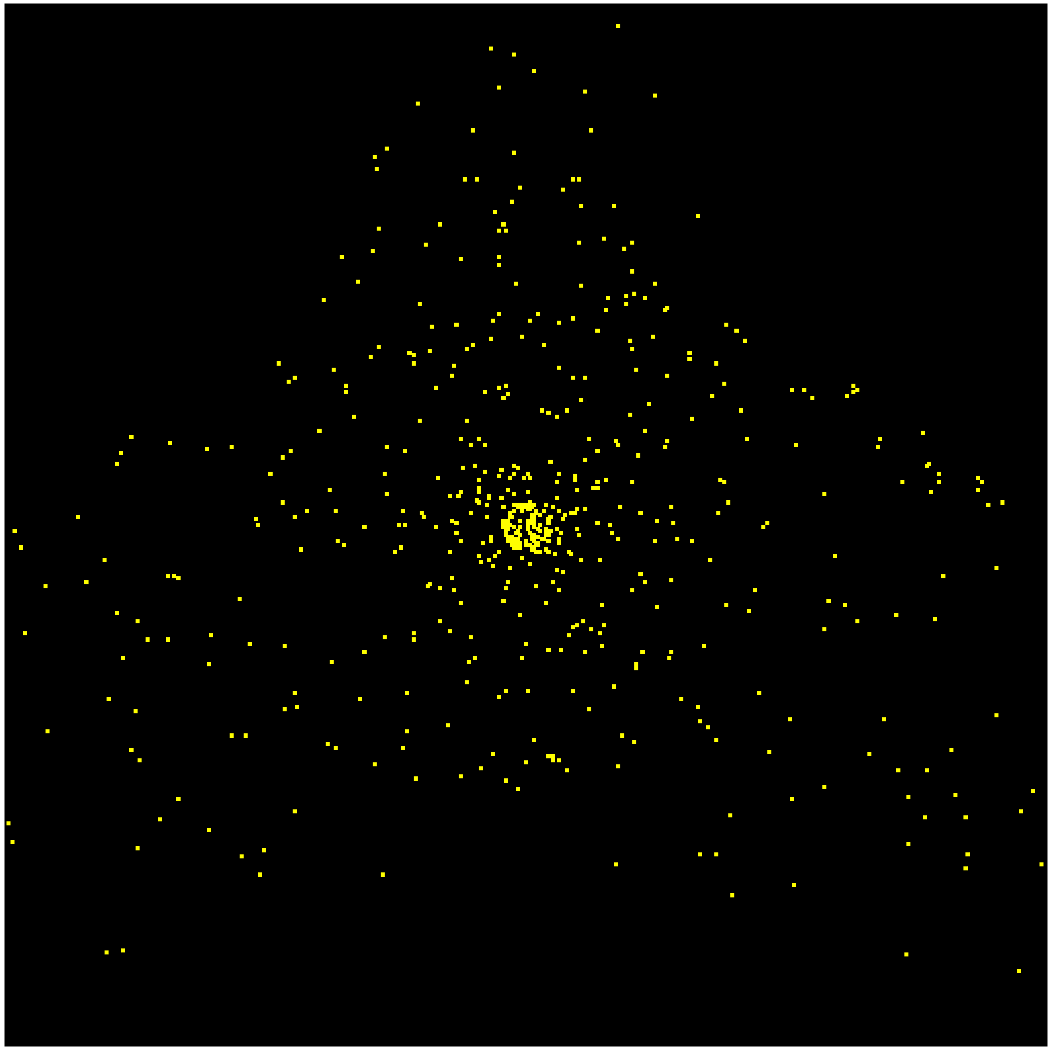


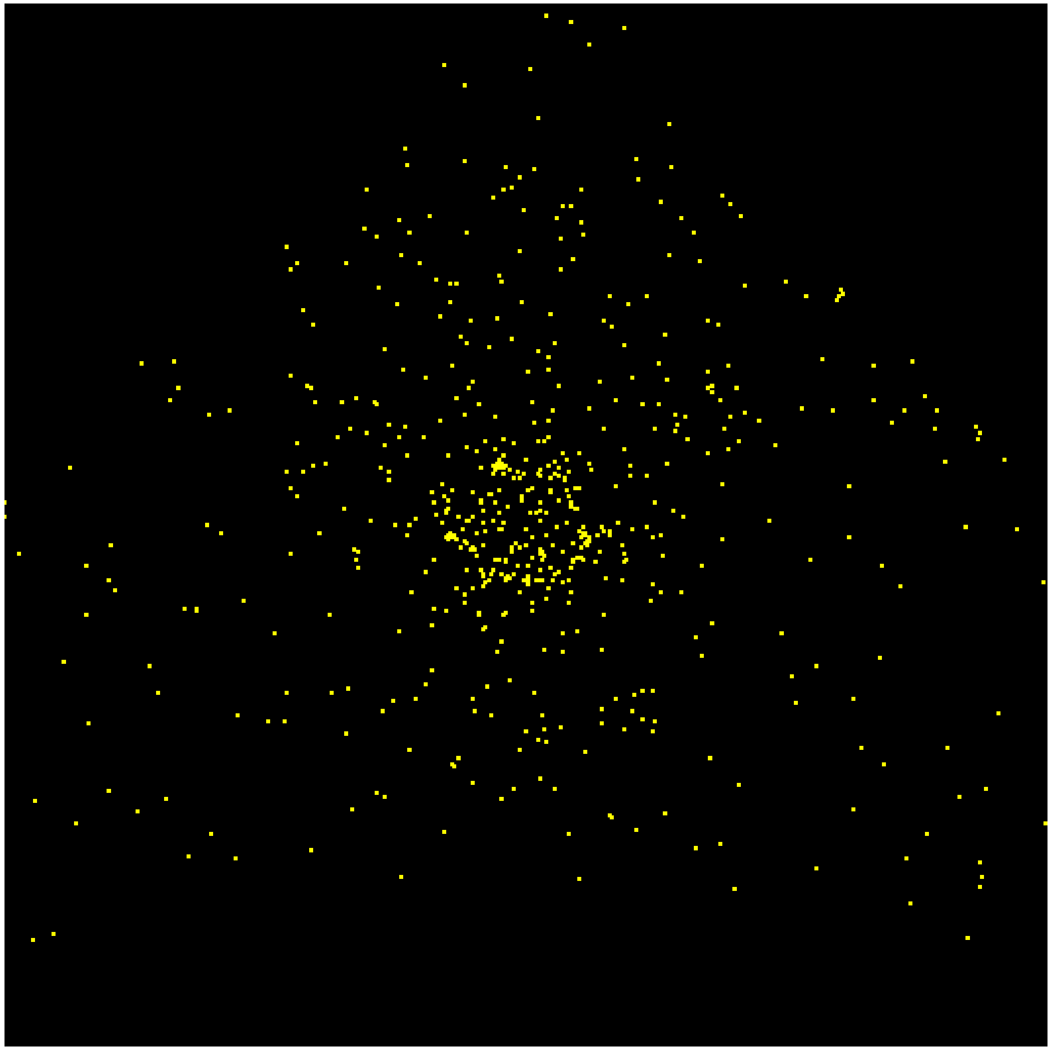


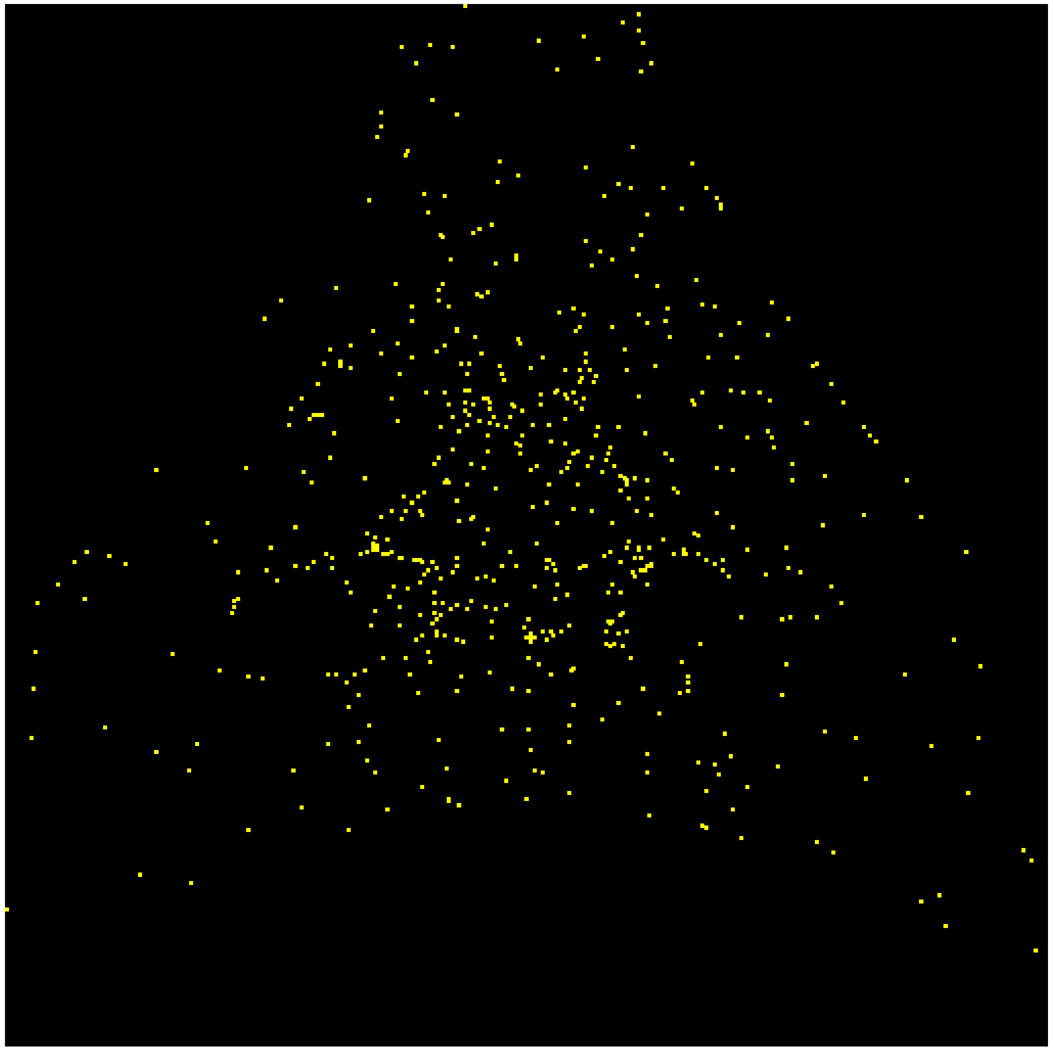


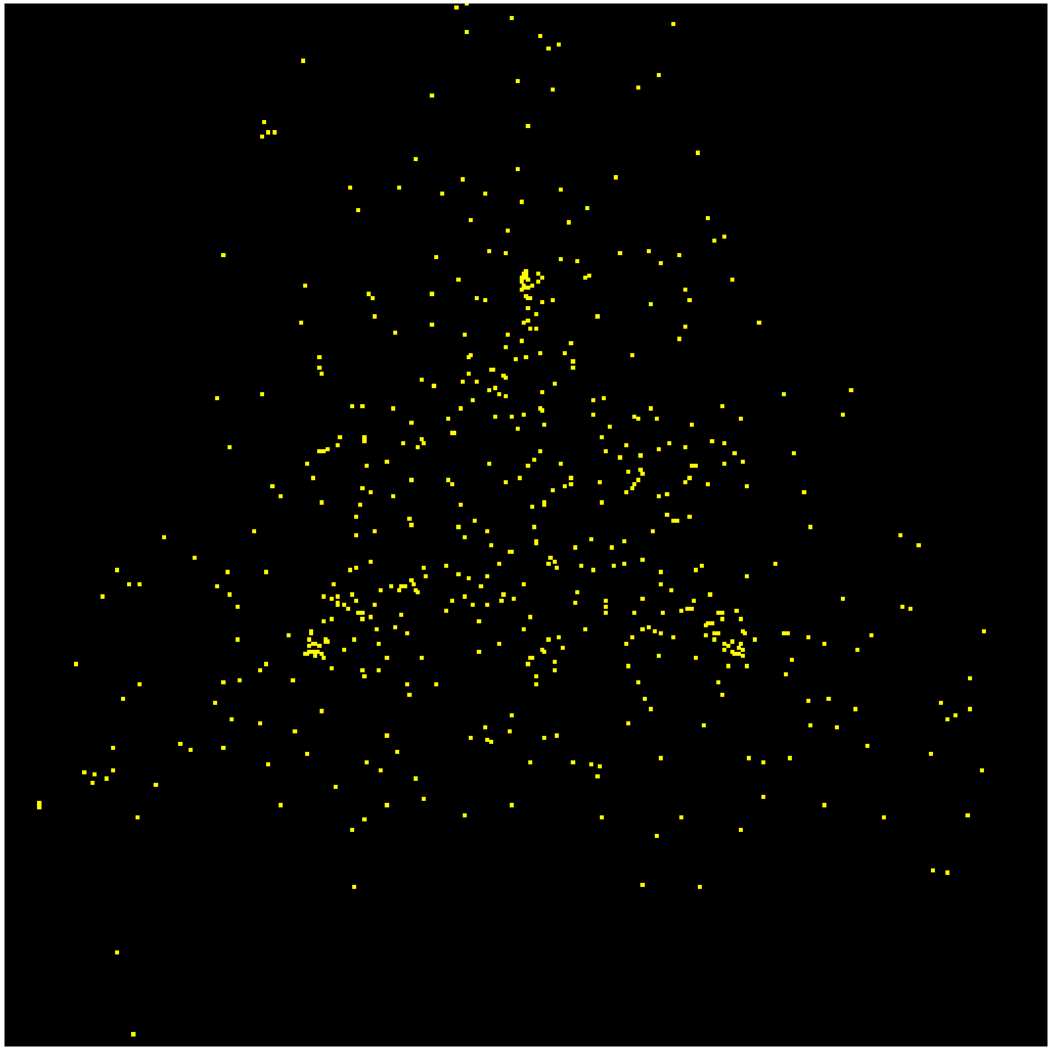


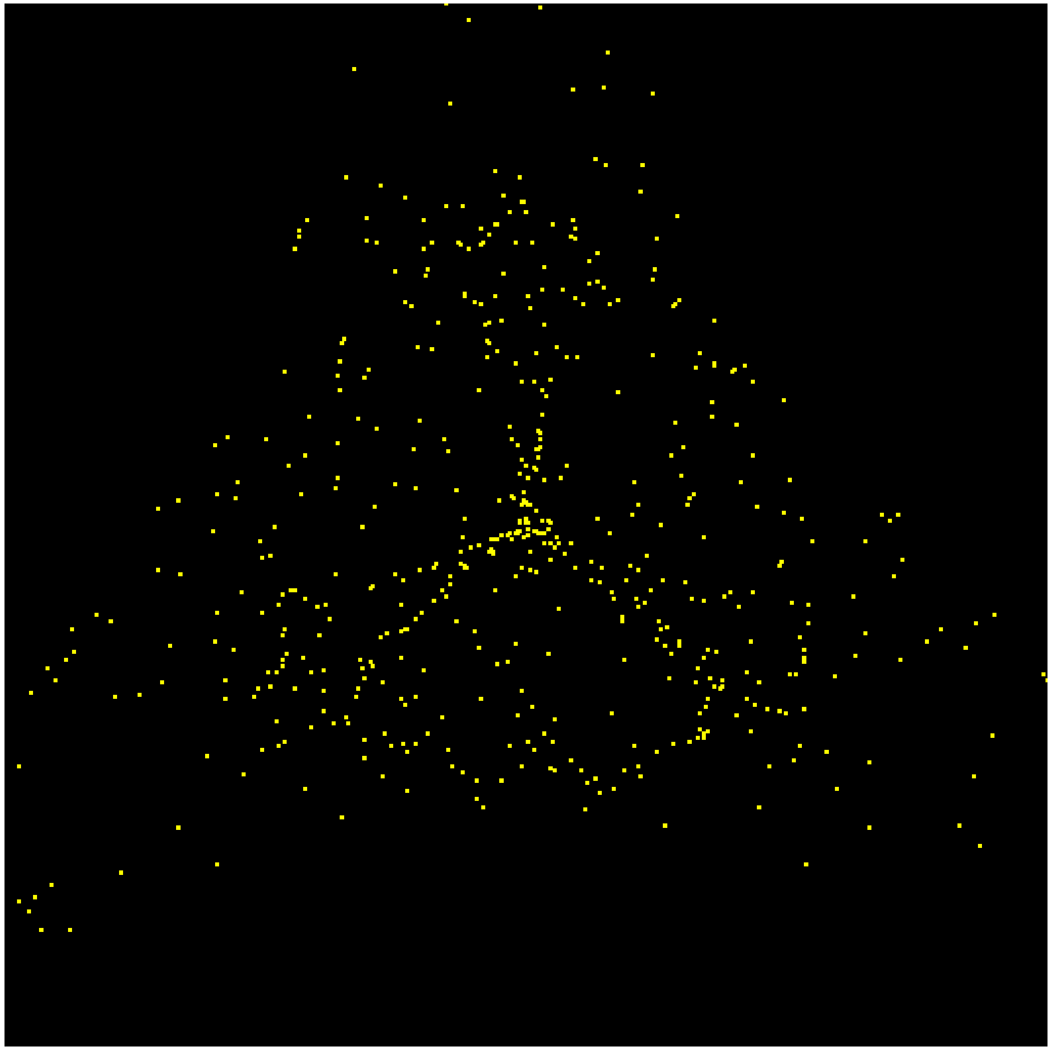












Вычислительные шейдеры — вычисления общего назначения

main.cpp

```
#include <GL/glew.h>
#include <GLFW/glfw3.h>

#include <stdio.h>
#include <malloc.h>

const unsigned int window_width  = 512;
const unsigned int window_height = 512;

void initGL();

GLuint* bufferID;

void initBuffers(GLuint*&);
void transformBuffers(GLuint*);
void outputBuffers(GLuint*);
```

```
int main(){

  initGL();

  bufferID=(GLuint*)calloc(2, sizeof(GLuint));

  initBuffers(bufferID);
  transformBuffers(bufferID);
  outputBuffers(bufferID);

  glDeleteBuffers(2,bufferID);
  free(bufferID);
  glfwTerminate();

  return 0;
}
```

```
void initGL(){
    GLFWwindow* window;

    if( !glfwInit() ){
        fprintf( stderr, "Failed to initialize GLFW\n" );
        getchar();
        return;
    }
    glfwWindowHint(GLFW_VISIBLE, 0);
    glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 4);
    glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
    glfwWindowHint(GLFW_OPENGL_FORWARD_COMPAT, GL_TRUE);
    glfwWindowHint(GLFW_OPENGL_PROFILE,
                                    GLFW_OPENGL_COMPAT_PROFILE);
    window = glfwCreateWindow( window_width, window_height,
                    "Template window", NULL, NULL);
    if( window == NULL ){
        fprintf( stderr, "Failed to open GLFW window. \n" );
        getchar();
        glfwTerminate();
        return;
    }
    glfwMakeContextCurrent(window);
    glewExperimental = true;
    if (glewInit() != GLEW_OK) {
        fprintf(stderr, "Failed to initialize GLEW\n");
        getchar();
        glfwTerminate();
        return;
    }

    return;
}
```

csh_common.cpp

```
#include <GL/glew.h>
#include <stdio.h>
#include <string>
#include <string.h>
#include <stdlib.h>

void checkErrors(std::string desc) {
    GLenum e = glGetError();
    if (e != GL_NO_ERROR) {
        fprintf(stderr, "OpenGL error in \"%s\": %s (%d)\n", desc.c_str(),
                                                    gluErrorString(e), e);
        exit(20);
    }
}

const int N=256;
```

```
GLuint genInitProg();
int initBuffers(GLuint*& bufferID){
  glGenBuffers(2, bufferID);

  glBindBuffer(GL_SHADER_STORAGE_BUFFER, bufferID[0]);
  glBufferData(GL_SHADER_STORAGE_BUFFER, N * sizeof(float), 0,
             GL_DYNAMIC_DRAW);

  glBindBuffer(GL_SHADER_STORAGE_BUFFER, bufferID[1]);
  glBufferData(GL_SHADER_STORAGE_BUFFER, N * sizeof(float), 0,
             GL_DYNAMIC_DRAW);

  glBindBufferBase(GL_SHADER_STORAGE_BUFFER, 0, bufferID[0]);
  glBindBufferBase(GL_SHADER_STORAGE_BUFFER, 1, bufferID[1]);

  GLuint csInitID=genInitProg();
  glUseProgram(csInitID);

  glDispatchCompute(N/128, 1, 1);
  glMemoryBarrier(GL_SHADER_STORAGE_BARRIER_BIT |
                        GL_BUFFER_UPDATE_BARRIER_BIT);
  glDeleteProgram(csInitID);
}
```

```
GLuint genInitProg(){
    GLuint progHandle = glCreateProgram();
    GLuint cs = glCreateShader(GL_COMPUTE_SHADER);

    const char *cpSrc[] = {
      "#version 430\n",
      "layout (local_size_x = 128, local_size_y = 1, local_size_z = 1) in; \
       layout(std430, binding = 0) buffer BufferA{float A[];};\
       layout(std430, binding = 1) buffer BufferB{float B[];};\
       void main() {\
        uint index = gl_GlobalInvocationID.x;\
        A[index]=0.1*float(index);\
        B[index]=0.2*float(index);\
       }"
      };

    glShaderSource(cs, 2, cpSrc, NULL);
.........................................................................................................
```

```
GLuint genTransformProg();
int transformBuffers(GLuint* bufferID){
  glBindBufferBase(GL_SHADER_STORAGE_BUFFER, 0, bufferID[0]);
  glBindBufferBase(GL_SHADER_STORAGE_BUFFER, 1, bufferID[1]);
  GLuint csTransformID=genTransformProg();

  glUseProgram(csTransformID);
  glDispatchCompute(N/128, 1, 1);
  glMemoryBarrier(GL_SHADER_STORAGE_BARRIER_BIT |
              GL_BUFFER_UPDATE_BARRIER_BIT);

  glDeleteProgram(csTransformID);
}
```

```
GLuint genTransformProg(){
    GLuint progHandle = glCreateProgram();
    GLuint cs = glCreateShader(GL_COMPUTE_SHADER);

    const char *cpSrc[] = {
      "#version 430\n",
      "layout (local_size_x = 128, local_size_y = 1, local_size_z = 1) in; \
       layout(std430, binding = 0) buffer BufferA{float A[];};\
       layout(std430, binding = 1) buffer BufferB{float B[];};\
       void main() {\
        uint index = gl_GlobalInvocationID.x;\
        A[index]=A[index]+B[index];\
       }"
      };

    glShaderSource(cs, 2, cpSrc, NULL);
…………………………………………………………………………………………
```

Транспонирование матриц (наивный вариант)

```
"#version 430\n",
"layout (local_size_x = 16, local_size_y = 16, local_size_z = 1) in; \
 layout(std430, binding = 0) buffer BufferA{float A[];};\
 layout(std430, binding = 1) buffer BufferB{float B[];};\
 void main() {\
  uint indexX = gl_GlobalInvocationID.x;\
  uint indexY = gl_GlobalInvocationID.y;\
  A[indexX+16*indexY]=float(indexX+16*indexY);\
 }"
```

```
"#version 430\n",
"layout (local_size_x = 16, local_size_y = 16, local_size_z = 1) in; \
 layout(std430, binding = 0) buffer BufferA{float A[];};\
 layout(std430, binding = 1) buffer BufferB{float B[];};\
 void main() {\
  uint indexX = gl_GlobalInvocationID.x;\
  uint indexY = gl_GlobalInvocationID.y;\
  B[indexX+16*indexY]=A[indexY+16*indexX];\
 }"
```

Экспорт данных из буфера на хост

```
void outputBuffers(GLuint* bufferID){
  glBindBuffer(GL_SHADER_STORAGE_BUFFER, bufferID[1]);
  float* data = (float*)glMapBuffer(GL_SHADER_STORAGE_BUFFER,
                                                        GL_READ_ONLY);

  float* hdata=(float*)calloc(N, sizeof(float));
  memcpy(&hdata[0], data, sizeof(float)*N);
  glUnmapBuffer(GL_SHADER_STORAGE_BUFFER);

  for(int i = 0; i < Nx; i++){
    for(int j = 0; j < Ny; j++)
      fprintf(stdout,"%g\t",hdata[j+i*Ny]);
    printf("\n");
  }
}
```

Встроенные типы переменных

```
vec2  a=vec2(1.0, 0.5);
vec2  b=vec2(2.0, -1.0);
vec4  c=vec4(vec2(0.9, 1.1), vec2(2.0, 0.5) );
vec3  d=vec3(vec2(0.9, 1.1),0.7);
vec4 e=vec4(vec3(vec2(0.9, 1.1),0.7), 29.0);

vec4 A;
A=vec4(a,b);

float F;
F=a[0];
F=a.x;
F=e.x;....F=e.w;
F=e[0];..F=e[3];

mat2 m,m1;
m=mat2(a,b);
m1=mat2(
      1.0,2.5,
      3.0,2.9
);
```

```
"#version 430\n",
"layout (local_size_x = 16, local_size_y = 16, local_size_z = 1) in; \
 layout(std430, binding = 0) buffer BufferA{float A[];};\
 layout(std430, binding = 1) buffer BufferB{float B[];};\
 void main() {\
  uint indexX = gl_GlobalInvocationID.x;\
  uint indexY = gl_GlobalInvocationID.y;\
  vec2 a=vec2(A[indexY+16*indexX], 0.5);\
  mat2 M;\
  M[0]=a;\
  M[1]=vec2(0.5,2.0);\
  vec2 ar=M*a;\
  vec3 b=vec3(a, 1.5);\
  vec3 c=b-vec3(0.0,0.0,0.5);\
  c=c.zyy; /*swizzle*/\
  float s;\
  s=dot(b,c);\
  B[indexX+16*indexY]=A[indexY+16*indexX]+s;\
}"
```

UNIFORM переменные

```
      uniform float coeff;\
      void main() {\
.............................................................................
       B[indexX+16*indexY]=A[indexY+16*indexX]+coeff;\
.............................................................................
```

```
..............................................................................................
 GLuint csTransformID=genTransformProg();

  glUseProgram(csTransformID);

  GLuint coeffID = glGetUniformLocation(csTransformID,"coeff");
  float c=2.5;
  glUniform1f(coeffID, c);
...................................................................
```

Спасибо за внимание!