

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ТЕЛЕКОММУНИКАЦИЯ И ИНФОРМАТИКИ»

КАФЕДРА ПМиК

Расчетно-графическая работа
по дисциплине «Программирование графических процессоров»

Выполнил: студент 3-го курса,
группы ИП-013
Копытина Т.А

Проверил: Старший преподаватель
Кафедры прикладной математики и кибернетики
Нужнов А.В.

Новосибирск, 2023

Оглавление

Постановка задачи.....	3
Описание архитектуры вычислительных шейдеров OpenGL.	4
Описание реализации.....	5
Скриншоты процесса работы программы	8
Анализ результатов программы.....	13
Листинг программы	15

Постановка задачи

Текст задания: *Сравнение производительности программы на основе Open GL - вычислительных шейдеров.*

Цель данного проекта – провести сравнительный анализ производительности программ, реализованных на алгоритмах линейной алгебры с использованием шейдеров Open GL.

- Изучить основные принципы работы вычислительных шейдеров OpenGL.
- Реализовать несколько алгоритмов линейной алгебры с использованием вычислительных шейдеров.
- Провести тестирование разработанных программ на различных объемах данных и сравнить полученные результаты в терминах времени выполнения операций линейной алгебры.
- Проанализировать полученные результаты и сделать выводы о производительности каждого подхода и эффективности использования вычислительных шейдеров OpenGL.

Для решения поставленных задач будут реализованы две программы на вычислительных шейдерах, которые выполняют операции линейной алгебры: умножение матрицы на вектор и транспонирование матрицы.

Описание архитектуры вычислительных шейдеров OpenGL.

Шейдеры — это небольшие программы, выполняемые на графическом ускорителе. Эти программы выполняются для каждого конкретного участка графического конвейера. Если описывать шейдеры наиболее простым способом, то шейдеры — это не более чем программы преобразующие входы в выходы. Шейдеры обычно изолированы друг от друга, и не имеют механизмов коммуникации между собой кроме упомянутых выше входов и выходов.

Вычислительные шейдеры работают совсем по-другому. "Пространство", с которым работает вычислительный шейдер, в значительной степени абстрактно; каждый вычислительный шейдер сам решает, что означает это пространство. Количество выполнений вычислительного шейдера определяется функцией, используемой для выполнения вычислительной операции. Самое важное из всего, что вычислительные шейдеры не имеют определяемых пользователем входных данных и вообще никаких выходных данных. Встроенные входные данные определяют только то, где в "пространстве" выполнения находится конкретный вызов вычислительного шейдера.

Описание реализации

Данная программа представляет собой реализацию операций над матрицами, таких как транспонирование и умножение матрицы на вектор.

Для проверки правильности транспонирования и умножения, программа включает шейдеры `transpose.comp` и `multiply.comp`, которые показывают результаты выполнения соответствующих операций.

Для транспонирования матрицы используется шейдер `transpose.comp`, в нем описывается следующее:

- `layout (local_size_x = 20, local_size_y = 20) in;` Задание размеров матрицы и точка входа в программу;
- `layout (std430, binding = 0) buffer InBuffer { float matrixA[];};`
`layout (std430, binding = 1) buffer OutBuffer {float matrixB[];};`
создание буферов под матрицы, чтобы копировать с CPU на GPU;
- Далее идет функция `void main()`, где выполняется само транспонирование.

Далее в файле `main.cpp` происходит вычислительный процесс и замер времени операции:

- В классе `class Transposing: public GPUAlgorithm` мы вызываем функцию `Transposing()`, в которой открываем наш шейдер *transpose.comp*, затем создаем новый шейдер внутри главной программы. Далее проверяем что компиляция шейдера прошла успешно. С помощью `CreateProgram()` создаем новую программу где будут выполняться все действия.
- Далее вызываем функцию `void GetResult()` где выводится результат работы функции транспонирования.
- Затем вызывается функция `void Transpose()` в ней уже проходит заполнение матрицы, работа шейдера, замер времени с помощью

`chrono::high_resolution_clock::time_point start = wtime()` копирование результата в буфер, затем очищение буфера.

Для умножения матрицы на вектор мы используем второй вычислительный шейдер `multiply.comp`, происходит следующее:

- Сначала задаем размеры матрицы `layout (local_size_x = 20, local_size_y = 20) in;`. Затем мы создаем три буфера на графическом процессоре для: матрицы, вектора и результата.
- Далее идет функция `void main()`, где выполняется умножение матрицы на вектор.

Далее в файле `main.cpp` происходит вычислительный процесс и замер времени операции:

- В классе `class Multiplication : public GPUAlgorithm` мы вызываем функцию `Multiplication()` в которой открываем шейдер `multiply.comp`, затем создаем новый шейдер внутри главной программы. Далее проверяем что компиляция шейдера прошла успешно. С помощью `CreateProgramm()` создаем новую программу где будут выполняться все действия.
- Далее вызываем функцию `void GetResult()` где выводится результат работы функции транспонирования.
- Затем вызывается функция `void Multiply()` в ней уже проходит заполнение матрицы, работа шейдера, замер времени с помощью `chrono::high_resolution_clock::time_point start = wtime()` копирование результата в буфер, затем очищение буфера.

Так же в основном файле `main.cpp` есть класс `class Controller`, в котором мы вызываем следующие функции:

- `Controller()`, `void InitGLFW()` с помощью этих функций мы создаем окно в которое мы выводим работу наших шейдеров и программ и их результаты.
- В функции `void Compute()` мы выводим результаты работы программ транспонирования и умножения матрицы на вектор.

Скриншоты процесса работы программы

```
--MATRIX--
 0  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 59
60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79
80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99
100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119
120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139
140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159
160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179
180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199
200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219
220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239
240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259
260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279
280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299
300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319
320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339
340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359
360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379
380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397 398 399
```

Рис.1 Матрица до преобразования размера 20x20.

```
--VECTOR--
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
```

Рис.2 Вектор размера 20x1

```
--RESULT--
2470 6270 10070 13870 17670 21470 25270 29070 32870 36670 40470 44270 48070 51870 55670 59470 63270 67070 70870 74670
Multiply Time (20 x 20) = 10 milliseconds
```

Рис.3 Результат умножения и замер времени данной операции.

```
--MATRIX--
 0  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 59
60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79
80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99
100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119
120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139
140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159
160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179
180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199
200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219
220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239
240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259
260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279
280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299
300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319
320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339
340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359
360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379
380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397 398 399
```

Рис.4 Матрица до преобразования размера 20x20.

--RESULT--																			
0	20	40	60	80	100	120	140	160	180	200	220	240	260	280	300	320	340	360	380
1	21	41	61	81	101	121	141	161	181	201	221	241	261	281	301	321	341	361	381
2	22	42	62	82	102	122	142	162	182	202	222	242	262	282	302	322	342	362	382
3	23	43	63	83	103	123	143	163	183	203	223	243	263	283	303	323	343	363	383
4	24	44	64	84	104	124	144	164	184	204	224	244	264	284	304	324	344	364	384
5	25	45	65	85	105	125	145	165	185	205	225	245	265	285	305	325	345	365	385
6	26	46	66	86	106	126	146	166	186	206	226	246	266	286	306	326	346	366	386
7	27	47	67	87	107	127	147	167	187	207	227	247	267	287	307	327	347	367	387
8	28	48	68	88	108	128	148	168	188	208	228	248	268	288	308	328	348	368	388
9	29	49	69	89	109	129	149	169	189	209	229	249	269	289	309	329	349	369	389
10	30	50	70	90	110	130	150	170	190	210	230	250	270	290	310	330	350	370	390
11	31	51	71	91	111	131	151	171	191	211	231	251	271	291	311	331	351	371	391
12	32	52	72	92	112	132	152	172	192	212	232	252	272	292	312	332	352	372	392
13	33	53	73	93	113	133	153	173	193	213	233	253	273	293	313	333	353	373	393
14	34	54	74	94	114	134	154	174	194	214	234	254	274	294	314	334	354	374	394
15	35	55	75	95	115	135	155	175	195	215	235	255	275	295	315	335	355	375	395
16	36	56	76	96	116	136	156	176	196	216	236	256	276	296	316	336	356	376	396
17	37	57	77	97	117	137	157	177	197	217	237	257	277	297	317	337	357	377	397
18	38	58	78	98	118	138	158	178	198	218	238	258	278	298	318	338	358	378	398
19	39	59	79	99	119	139	159	179	199	219	239	259	279	299	319	339	359	379	399

Рис.5 Результат: транспонированная матрица

```
Transpose Matrix Time (20 x 20) = 75 milliseconds
```

Рис.6 Замер времени транспонирования.

```
--MATRIX--
 0  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 59
60 61 62 63 64 65 66 67 68 69
70 71 72 73 74 75 76 77 78 79
80 81 82 83 84 85 86 87 88 89
90 91 92 93 94 95 96 97 98 99

--VECTOR--
0 1 2 3 4 5 6 7 8 9

--RESULT--
285 735 1185 1635 2085 2535 2985 3435 3885 4335
Multiply Time (10 x 10) = 8 milliseconds
```

Рис7. Матрица размера 10x10, вектор размера 10x1, результат умножения и замер времени.

```
--MATRIX--
 0  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 59
60 61 62 63 64 65 66 67 68 69
70 71 72 73 74 75 76 77 78 79
80 81 82 83 84 85 86 87 88 89
90 91 92 93 94 95 96 97 98 99

--RESULT--
 0 10 20 30 40 50 60 70 80 90
 1 11 21 31 41 51 61 71 81 91
 2 12 22 32 42 52 62 72 82 92
 3 13 23 33 43 53 63 73 83 93
 4 14 24 34 44 54 64 74 84 94
 5 15 25 35 45 55 65 75 85 95
 6 16 26 36 46 56 66 76 86 96
 7 17 27 37 47 57 67 77 87 97
 8 18 28 38 48 58 68 78 88 98
 9 19 29 39 49 59 69 79 89 99
Transpose Matrix Time (10 x 10) = 30 milliseconds
```

Рис.8 Матрица размера 10x10, транспонированная матрица размера 10x10 и замер времени выполнения операции.

```
--MATRIX--
 0  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 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74
75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99
100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124
125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149
150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174
175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199
200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224
225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249
250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274
275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299
300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324
325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349
350 351 352 353 354 355 356 357 358 359 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374
375 376 377 378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397 398 399
400 401 402 403 404 405 406 407 408 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423 424
425 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449
450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467 468 469 470 471 472 473 474
475 476 477 478 479 480 481 482 483 484 485 486 487 488 489 490 491 492 493 494 495 496 497 498 499
500 501 502 503 504 505 506 507 508 509 510 511 512 513 514 515 516 517 518 519 520 521 522 523 524
525 526 527 528 529 530 531 532 533 534 535 536 537 538 539 540 541 542 543 544 545 546 547 548 549
550 551 552 553 554 555 556 557 558 559 560 561 562 563 564 565 566 567 568 569 570 571 572 573 574
575 576 577 578 579 580 581 582 583 584 585 586 587 588 589 590 591 592 593 594 595 596 597 598 599
600 601 602 603 604 605 606 607 608 609 610 611 612 613 614 615 616 617 618 619 620 621 622 623 624

--VECTOR--
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
```

Рис.9 Матрица размером 25x25, вектор размером 25x1

```
--RESULT--
4900 12400 19900 27400 34900 42400 49900 57400 64900 72400 79900 87400 94900 102400 109900 117400 124900 132400 139900 147400 154900 162400 169900 177400 184900
Multiply Time (25 x 25) = 31 milliseconds
```

Рис.10 результат умножения матрицы на вектор и замер времени.

```
--MATRIX--
0 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 59
60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89
90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119
120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149
150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179
180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209
210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239
240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269
270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299
300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329
330 331 332 333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359
360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387 388 389
390 391 392 393 394 395 396 397 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412 413 414 415 416 417 418 419
420 421 422 423 424 425 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449
450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467 468 469 470 471 472 473 474 475 476 477 478 479
480 481 482 483 484 485 486 487 488 489 490 491 492 493 494 495 496 497 498 499 500 501 502 503 504 505 506 507 508 509
510 511 512 513 514 515 516 517 518 519 520 521 522 523 524 525 526 527 528 529 530 531 532 533 534 535 536 537 538 539
540 541 542 543 544 545 546 547 548 549 550 551 552 553 554 555 556 557 558 559 560 561 562 563 564 565 566 567 568 569
570 571 572 573 574 575 576 577 578 579 580 581 582 583 584 585 586 587 588 589 590 591 592 593 594 595 596 597 598 599
600 601 602 603 604 605 606 607 608 609 610 611 612 613 614 615 616 617 618 619 620 621 622 623 624 625 626 627 628 629
630 631 632 633 634 635 636 637 638 639 640 641 642 643 644 645 646 647 648 649 650 651 652 653 654 655 656 657 658 659
660 661 662 663 664 665 666 667 668 669 670 671 672 673 674 675 676 677 678 679 680 681 682 683 684 685 686 687 688 689
690 691 692 693 694 695 696 697 698 699 700 701 702 703 704 705 706 707 708 709 710 711 712 713 714 715 716 717 718 719
720 721 722 723 724 725 726 727 728 729 730 731 732 733 734 735 736 737 738 739 740 741 742 743 744 745 746 747 748 749
750 751 752 753 754 755 756 757 758 759 760 761 762 763 764 765 766 767 768 769 770 771 772 773 774 775 776 777 778 779
780 781 782 783 784 785 786 787 788 789 790 791 792 793 794 795 796 797 798 799 800 801 802 803 804 805 806 807 808 809
810 811 812 813 814 815 816 817 818 819 820 821 822 823 824 825 826 827 828 829 830 831 832 833 834 835 836 837 838 839
840 841 842 843 844 845 846 847 848 849 850 851 852 853 854 855 856 857 858 859 860 861 862 863 864 865 866 867 868 869
870 871 872 873 874 875 876 877 878 879 880 881 882 883 884 885 886 887 888 889 890 891 892 893 894 895 896 897 898 899

--VECTOR--
0 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
```

Рис.11 матрица размером 30x30, вектор размером 30x1

```
--RESULT--
8555 21605 34655 47705 60755 73805 86855 99905 112955 126005 139055 152105 165155 178205 191255 204305 217355 230405
243455 256505 269555 282605 295655 308705 321755 334805 347855 360905 373955 387005
Multiply Time (30 x 30) = 14 milliseconds
```

Рис.12 Результат умножения и замер времени.

```
--MATRIX--
0 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 59
60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89
90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119
120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149
150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179
180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209
210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239
240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269
270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299
300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329
330 331 332 333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359
360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387 388 389
390 391 392 393 394 395 396 397 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412 413 414 415 416 417 418 419
420 421 422 423 424 425 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449
450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467 468 469 470 471 472 473 474 475 476 477 478 479
480 481 482 483 484 485 486 487 488 489 490 491 492 493 494 495 496 497 498 499 500 501 502 503 504 505 506 507 508 509
510 511 512 513 514 515 516 517 518 519 520 521 522 523 524 525 526 527 528 529 530 531 532 533 534 535 536 537 538 539
540 541 542 543 544 545 546 547 548 549 550 551 552 553 554 555 556 557 558 559 560 561 562 563 564 565 566 567 568 569
570 571 572 573 574 575 576 577 578 579 580 581 582 583 584 585 586 587 588 589 590 591 592 593 594 595 596 597 598 599
600 601 602 603 604 605 606 607 608 609 610 611 612 613 614 615 616 617 618 619 620 621 622 623 624 625 626 627 628 629
630 631 632 633 634 635 636 637 638 639 640 641 642 643 644 645 646 647 648 649 650 651 652 653 654 655 656 657 658 659
660 661 662 663 664 665 666 667 668 669 670 671 672 673 674 675 676 677 678 679 680 681 682 683 684 685 686 687 688 689
690 691 692 693 694 695 696 697 698 699 700 701 702 703 704 705 706 707 708 709 710 711 712 713 714 715 716 717 718 719
720 721 722 723 724 725 726 727 728 729 730 731 732 733 734 735 736 737 738 739 740 741 742 743 744 745 746 747 748 749
750 751 752 753 754 755 756 757 758 759 760 761 762 763 764 765 766 767 768 769 770 771 772 773 774 775 776 777 778 779
780 781 782 783 784 785 786 787 788 789 790 791 792 793 794 795 796 797 798 799 800 801 802 803 804 805 806 807 808 809
810 811 812 813 814 815 816 817 818 819 820 821 822 823 824 825 826 827 828 829 830 831 832 833 834 835 836 837 838 839
840 841 842 843 844 845 846 847 848 849 850 851 852 853 854 855 856 857 858 859 860 861 862 863 864 865 866 867 868 869
870 871 872 873 874 875 876 877 878 879 880 881 882 883 884 885 886 887 888 889 890 891 892 893 894 895 896 897 898 899
```

Рис.13 Матрица до изменения размера 30x30

```
--RESULT--
0 30 60 90 120 150 180 210 240 270 300 330 360 390 420 450 480 510 540 570 600 630 660 690 720 750 780 810 840 870
1 31 61 91 121 151 181 211 241 271 301 331 361 391 421 451 481 511 541 571 601 631 661 691 721 751 781 811 841 871
2 32 62 92 122 152 182 212 242 272 302 332 362 392 422 452 482 512 542 572 602 632 662 692 722 752 782 812 842 872
3 33 63 93 123 153 183 213 243 273 303 333 363 393 423 453 483 513 543 573 603 633 663 693 723 753 783 813 843 873
4 34 64 94 124 154 184 214 244 274 304 334 364 394 424 454 484 514 544 574 604 634 664 694 724 754 784 814 844 874
5 35 65 95 125 155 185 215 245 275 305 335 365 395 425 455 485 515 545 575 605 635 665 695 725 755 785 815 845 875
6 36 66 96 126 156 186 216 246 276 306 336 366 396 426 456 486 516 546 576 606 636 666 696 726 756 786 816 846 876
7 37 67 97 127 157 187 217 247 277 307 337 367 397 427 457 487 517 547 577 607 637 667 697 727 757 787 817 847 877
8 38 68 98 128 158 188 218 248 278 308 338 368 398 428 458 488 518 548 578 608 638 668 698 728 758 788 818 848 878
9 39 69 99 129 159 189 219 249 279 309 339 369 399 429 459 489 519 549 579 609 639 669 699 729 759 789 819 849 879
10 40 70 100 130 160 190 220 250 280 310 340 370 400 430 460 490 520 550 580 610 640 670 700 730 760 790 820 850 880
11 41 71 101 131 161 191 221 251 281 311 341 371 401 431 461 491 521 551 581 611 641 671 701 731 761 791 821 851 881
12 42 72 102 132 162 192 222 252 282 312 342 372 402 432 462 492 522 552 582 612 642 672 702 732 762 792 822 852 882
13 43 73 103 133 163 193 223 253 283 313 343 373 403 433 463 493 523 553 583 613 643 673 703 733 763 793 823 853 883
14 44 74 104 134 164 194 224 254 284 314 344 374 404 434 464 494 524 554 584 614 644 674 704 734 764 794 824 854 884
15 45 75 105 135 165 195 225 255 285 315 345 375 405 435 465 495 525 555 585 615 645 675 705 735 765 795 825 855 885
16 46 76 106 136 166 196 226 256 286 316 346 376 406 436 466 496 526 556 586 616 646 676 706 736 766 796 826 856 886
17 47 77 107 137 167 197 227 257 287 317 347 377 407 437 467 497 527 557 587 617 647 677 707 737 767 797 827 857 887
18 48 78 108 138 168 198 228 258 288 318 348 378 408 438 468 498 528 558 588 618 648 678 708 738 768 798 828 858 888
19 49 79 109 139 169 199 229 259 289 319 349 379 409 439 469 499 529 559 589 619 649 679 709 739 769 799 829 859 889
20 50 80 110 140 170 200 230 260 290 320 350 380 410 440 470 500 530 560 590 620 650 680 710 740 770 800 830 860 890
21 51 81 111 141 171 201 231 261 291 321 351 381 411 441 471 501 531 561 591 621 651 681 711 741 771 801 831 861 891
22 52 82 112 142 172 202 232 262 292 322 352 382 412 442 472 502 532 562 592 622 652 682 712 742 772 802 832 862 892
23 53 83 113 143 173 203 233 263 293 323 353 383 413 443 473 503 533 563 593 623 653 683 713 743 773 803 833 863 893
24 54 84 114 144 174 204 234 264 294 324 354 384 414 444 474 504 534 564 594 624 654 684 714 744 774 804 834 864 894
25 55 85 115 145 175 205 235 265 295 325 355 385 415 445 475 505 535 565 595 625 655 685 715 745 775 805 835 865 895
26 56 86 116 146 176 206 236 266 296 326 356 386 416 446 476 506 536 566 596 626 656 686 716 746 776 806 836 866 896
27 57 87 117 147 177 207 237 267 297 327 357 387 417 447 477 507 537 567 597 627 657 687 717 747 777 807 837 867 897
28 58 88 118 148 178 208 238 268 298 328 358 388 418 448 478 508 538 568 598 628 658 688 718 748 778 808 838 868 898
29 59 89 119 149 179 209 239 269 299 329 359 389 419 449 479 509 539 569 599 629 659 689 719 749 779 809 839 869 899
Transpose Matrix Time (30 x 30) = 193 milliseconds
```

рис.14 Матрица после транспонирования размера 30x30 и замер времени выполнения операции.

Анализ результатов программы

Таблица результатов для разной размерности матрицы:

	Транспонирование	Умножение матрицы на вектор
матрица 10x10	30	8
матрица 20x20	75	10
матрица 25x25	122	31
матрица 30x30	193	14

Рис.15 Таблица результатов транспонирования и умножения матрицы на вектор.

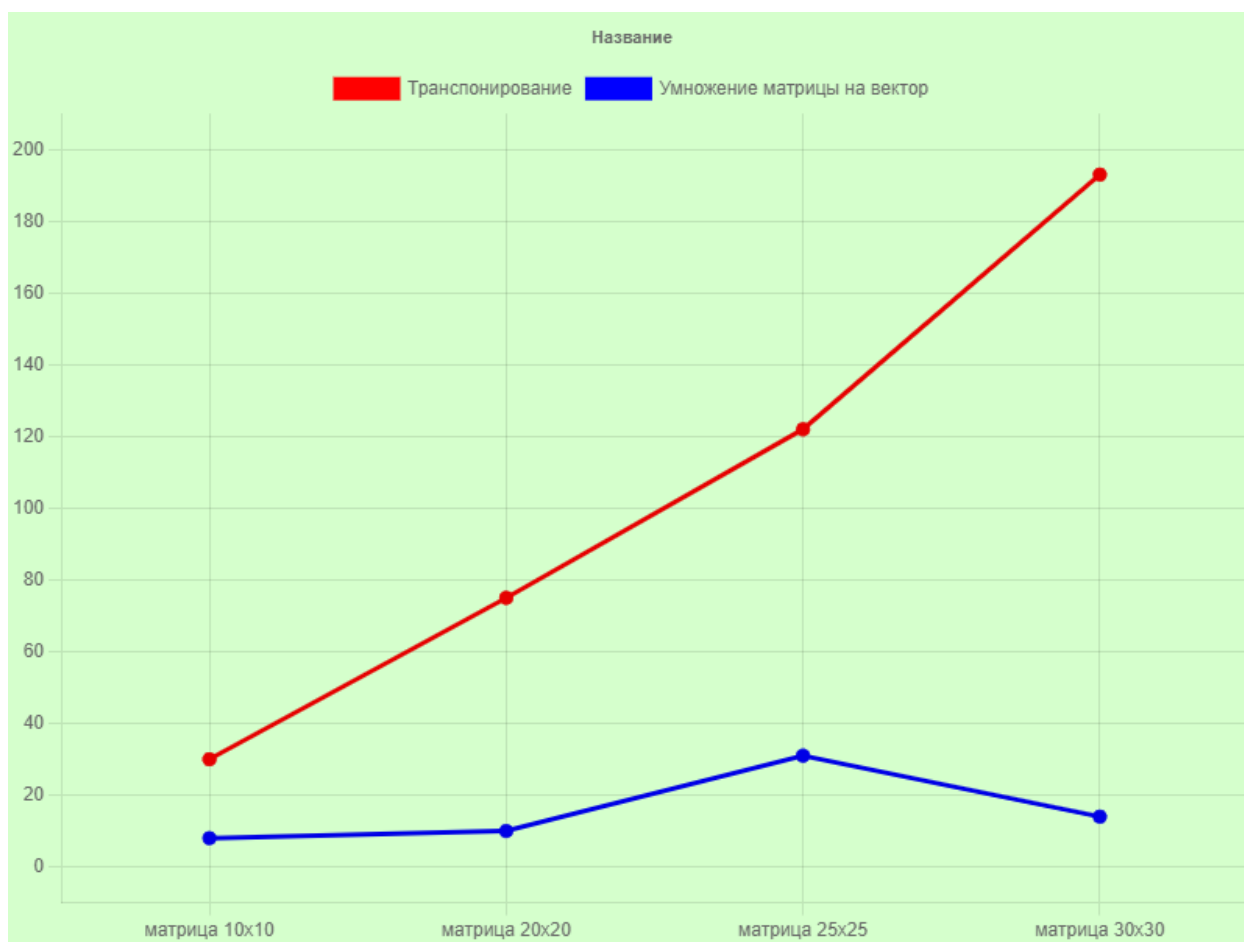


рис.16 Сравнение результатов транспонирования и умножения.

Результаты показывают, что использование вычислительных шейдеров OpenGL помогают значительно ускорить выполнение операций с матрицами на GPU.

В целом использование специализированных библиотек таких как OpenGL, может значительно упростить и ускорить разработку и выполнение с матрицами на GPU. При выборе инструментов для работы с матрицами стоит учитывать размеры матриц и тип операций, которые нужно выполнить.

Листинг программы

transpose.comp

```
#version 430

layout (local_size_x = 30, local_size_y = 30) in;
layout (std430, binding = 0) buffer InBuffer { float matrixA[];};
layout (std430, binding = 1) buffer OutBuffer {float matrixB[];};

void main()
{
    uint glindex = gl_GlobalInvocationID.y * gl_WorkGroupSize.x +
gl_GlobalInvocationID.x;

    uint lengthLine = 30;
    uint i = glindex / lengthLine;
    uint j = glindex % lengthLine;

    uint newIndex = j * lengthLine + i;
    matrixB[newIndex] = matrixA[glindex];
}
```

multiply.comp

```
#version 430
```

```
layout (local_size_x = 30, local_size_y = 30) in;
```

```
layout (std430, binding = 0) buffer MatrixBuffer {  
    float matrix[];  
};
```

```
layout (std430, binding = 1) buffer VectorBuffer {  
    float vector[];  
};
```

```
layout (std430, binding = 2) buffer ResultBuffer {  
    float result[];  
};
```

```
uniform uint matrixRows;  
uniform uint matrixColumns;
```

```
void main() {  
    uint globalIndex = gl_GlobalInvocationID.y * gl_WorkGroupSize.x +  
gl_GlobalInvocationID.x;  
  
    uint rowIndex = globalIndex / matrixColumns;  
  
    float dotProduct = 0.0;  
    for (uint i = 0; i < matrixColumns; i++) {  
        float matrixValue = matrix[rowIndex * matrixColumns + i];  
        float vectorValue = vector[i];  
        dotProduct += matrixValue * vectorValue;  
    }  
  
    result[rowIndex] = dotProduct;  
}
```


Main.cpp

```
#include <GL/glew.h>
#include <cstdlib>
#include <iostream>
#include <sstream>
#include <fstream>
#include <vector>
#include <ctime>
#include <GLFW/glfw3.h>
#include <GL/gl.h>
#include <chrono>
#include <iomanip>
using namespace std;

chrono::high_resolution_clock::time_point wtime() {
    return chrono::high_resolution_clock::now();
}

void errorCallback(int error, const char* description) {
    cerr << "GLFW ERROR " << error << ": " << description << endl;
}

class GPUAlgorithm
{
protected:
    const char* shaderString;
    GLuint shader;
    GLint compileTrue;
    GLuint program;
    GLuint N;
    double time;
    GLuint matrixBuffer;
    GLuint vectorBuffer;
    GLuint resultBuffer;
public:
```

```

void SetSize(GLuint N)
{
    this->N = N;
}

void CreateProgramm()
{
    program = glCreateProgram();
    glAttachShader(program, shader);
    glLinkProgram(program);
}

void InitMatrix(vector<float>& matrix)
{
    for (GLuint i = 0; i < N * N; i++) {
        matrix.push_back(i);
    }
}

void InitVector(vector<float>& vector)
{
    for (GLuint i = 0; i < N; i++) {
        vector.push_back(i);
    }
}

void PrintMatrix(vector<float>& matrix)
{
    cout << "--MATRIX--\n";
    for (int i = 0; i < N * N; ++i) {
        cout << setw(4) << matrix[i] << ' ';
        if (i > 0 && i % N == N - 1) cout << endl;
    }
}

void PrintVector(vector<float>& vector)
{
    cout << "\n--VECTOR--\n";
    for (int i = 0; i < N; ++i) {
        cout << vector[i] << ' ';
    }
}

```

```

        }
        cout << endl;
    }
    double GetTime()
    {
        return time;
    }
};

class Transposing: public GPUAlgorithm
{
private:
public:
    Transposing()
    {
        ifstream shaderFiles("transpose.comp");
        string shaderFile((istreambuf_iterator<char>(shaderFiles)),
istreambuf_iterator<char>());
        shaderString = shaderFile.c_str();
        shader = glCreateShader(GL_COMPUTE_SHADER);
        glShaderSource(shader, 1, &shaderString, nullptr);
        glCompileShader(shader)

;

        glGetShaderiv(shader, GL_COMPILE_STATUS, &compileTrue);

        if (!compileTrue)
        {
            cout << "Transposing Shader ERROR" << endl;
            glDeleteShader(shader);
        }
        CreateProgramm();
    }
    void GetResult()
    {
        float* resultData = new float[N * N];

```

```

        glBindBuffer(GL_SHADER_STORAGE_BUFFER, resultBuffer);
        glGetBufferSubData(GL_SHADER_STORAGE_BUFFER, 0, N * N *
sizeof(float), resultData);

        cout << "--RESULT--\n";
        for (int i = 0; i < N * N; ++i) {
            cout << setw(4) << resultData[i] << ' ';
            if (i > 0 && i % N == N - 1) cout << endl;
        }

        delete[] resultData;
    }
    void Transpose()
    {
        vector<float> matrixA;
        InitMatrix(matrixA);
        PrintMatrix(matrixA);
        cout << endl;

        chrono::high_resolution_clock::time_point start = wtime();

        GLuint matrixBufferBinding = 0;
        GLuint resultBufferBinding = 1;

        glGenBuffers(1, &matrixBuffer);
        glBindBuffer(GL_SHADER_STORAGE_BUFFER, matrixBuffer);
        glBufferData(GL_SHADER_STORAGE_BUFFER, N * N * sizeof(float),
matrixA.data(), GL_DYNAMIC_COPY);
        glBindBufferBase(GL_SHADER_STORAGE_BUFFER, 0, matrixBuffer);

        glGenBuffers(1, &resultBuffer);
        glBindBuffer(GL_SHADER_STORAGE_BUFFER, resultBuffer);
        glBufferData(GL_SHADER_STORAGE_BUFFER, N * N * sizeof(float),
nullptr, GL_DYNAMIC_COPY);
        glBindBufferBase(GL_SHADER_STORAGE_BUFFER, 1, resultBuffer);
        glUseProgram(program);
    }

```

```

        glDispatchCompute(1, 1, 1);
        glMemoryBarrier(GL_SHADER_STORAGE_BARRIER_BIT);
        GetResult();

        glDeleteBuffers(1, &matrixBuffer);
        glDeleteBuffers(1, &vectorBuffer);
        glDeleteBuffers(1, &resultBuffer);

        chrono::high_resolution_clock::time_point end = wtime();
        chrono::milliseconds timer =
std::chrono::duration_cast<chrono::milliseconds>(end - start);
        time = timer.count();

        glDeleteProgram(program);
        glDeleteShader(shader);
    }
};

class Multiplication : public GPUAlgorithm
{
private:
public:
    Multiplication()
    {
        ifstream shaderFiles("multiply.comp");
        string shaderFile((istreambuf_iterator<char>(shaderFiles)),
istreambuf_iterator<char>());
        shaderString = shaderFile.c_str();
        shader = glCreateShader(GL_COMPUTE_SHADER);
        glShaderSource(shader, 1, &shaderString, nullptr);
        glCompileShader(shader);
        glGetShaderiv(shader, GL_COMPILE_STATUS, &compileTrue);

        if (!compileTrue)
        {
            cout << "Multiplication Shader ERROR" << endl;
            glDeleteShader(shader);

```

```

    }

    CreateProgramm();
}

void GetResult()
{
    float* resultData = new float[N];

    glBindBuffer(GL_SHADER_STORAGE_BUFFER, resultBuffer);
    glGetBufferSubData(GL_SHADER_STORAGE_BUFFER, 0, N * sizeof(float),
resultData);

    cout << "--RESULT--\n";
    for (int i = 0; i < N; i++) {
        cout << resultData[i] << " ";
    }
    delete[] resultData;
}

void Multiply()
{
    vector<float> matrix;
    vector<float> vector;

    InitMatrix(matrix);

    InitVector(vector);

    PrintMatrix(matrix);
    PrintVector(vector);
    cout << endl;

    chrono::high_resolution_clock::time_point start = wtime();

    GLuint matrixBufferBinding = 0;
    GLuint vectorBufferBinding = 1;
    GLuint resultBufferBinding = 2;

```

```

        glGenBuffers(1, &matrixBuffer);
        glBindBuffer(GL_SHADER_STORAGE_BUFFER, matrixBuffer);
        glBufferData(GL_SHADER_STORAGE_BUFFER, N * N * sizeof(float),
matrix.data(), GL_DYNAMIC_COPY);
        glBindBufferBase(GL_SHADER_STORAGE_BUFFER, 0, matrixBuffer);

        glGenBuffers(1, &vectorBuffer);
        glBindBuffer(GL_SHADER_STORAGE_BUFFER, vectorBuffer);
        glBufferData(GL_SHADER_STORAGE_BUFFER, N * sizeof(float),
vector.data(), GL_DYNAMIC_COPY);
        glBindBufferBase(GL_SHADER_STORAGE_BUFFER, 1, vectorBuffer);

        glGenBuffers(1, &resultBuffer);
        glBindBuffer(GL_SHADER_STORAGE_BUFFER, resultBuffer);
        glBufferData(GL_SHADER_STORAGE_BUFFER, N * N * sizeof(float),
nullptr, GL_DYNAMIC_COPY);
        glBindBufferBase(GL_SHADER_STORAGE_BUFFER, 2, resultBuffer);


        GLuint matrixRowsLocation = glGetUniformLocation(program,
"matrixRows");
        GLuint matrixColumnsLocation = glGetUniformLocation(program,
"matrixColumns");

        glUseProgram(program);
        glUniform1ui(matrixRowsLocation, N);
        glUniform1ui(matrixColumnsLocation, N);

        glDispatchCompute(N, N, 1);

        glMemoryBarrier(GL_SHADER_STORAGE_BARRIER_BIT);

        GetResult();

        glDeleteBuffers(1, &matrixBuffer);

```

```

        glDeleteBuffers(1, &vectorBuffer);
        glDeleteBuffers(1, &resultBuffer);

        chrono::high_resolution_clock::time_point end = wtime();
        chrono::milliseconds timer =
std::chrono::duration_cast<chrono::milliseconds>(end - start);
        time = timer.count();

        glDeleteProgram(program);
        glDeleteShader(shader);
    }
};

class Controller
{
private:
    GLFWwindow* window;
    GLuint size = 30;
    GLuint w = 600;
    GLuint h = 600;
public:
    Controller()
    {
        InitGLFW();
        glewInit();
        glViewport(0, 0, w, h);
    }
    void InitGLFW()
    {
        glfwSetErrorCallback(errorCallback);
        glfwInit();
        glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 4);
        glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
        glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);
        glfwWindowHint(GLFW_RESIZABLE, GL_FALSE);
        window = glfwCreateWindow(w, h, "OpenGL", nullptr, nullptr);
    }
};

```



```

        glfwMakeContextCurrent(window);
        glewExperimental = GL_TRUE;
    }

    void Compute()
    {
        Multiplication multiplicationObject;
        multiplicationObject.SetSize(size);
        multiplicationObject.Multiply();

        cout << "\nMultiply Time (30 x 30) = " <<
multiplicationObject.GetTime() << " milliseconds" << endl;

        Transposing transposingObject;
        transposingObject.SetSize(size);
        transposingObject.Transpose();

        cout << "\nTranspose Matrix Time (30 x 30) = " <<
transposingObject.GetTime() << " milliseconds" << endl;

        glfwTerminate();
    }
};

int main()
{
    Controller controller;
    controller.Compute();
    return 0;
}

```