МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Тульский государственный университет»

Институт прикладной математики и компьютерных наук
Кафедра «Прикладная математика и информатика»

Утверждено на заседании кафедры
«Прикладная математика и информатика»
14 января 2020 г., протокол № 6

Заведующий кафедрой

_____ В.И. Иванов

МЕТОДИЧЕСКИЕ УКАЗАНИЯ
по выполнению лабораторных работ по дисциплине (модулю) «Компьютерная графика»

основной профессиональной образовательной программы высшего образования – программы бакалавриата

по направлению подготовки
01.03.02 Прикладная математика и информатика

с направленностью (профилем)
Прикладная математика и информатика

Форма обучения: очная

Идентификационный номер образовательной программы: 010302-01-20

Тула 2020 год

Разработчик методических указаний

Горбачев Д.В., профессор каф. ПМиИ, д.ф.-м.н. _____

(ФИО, должность, ученая степень, ученое звание) *(подпись)*

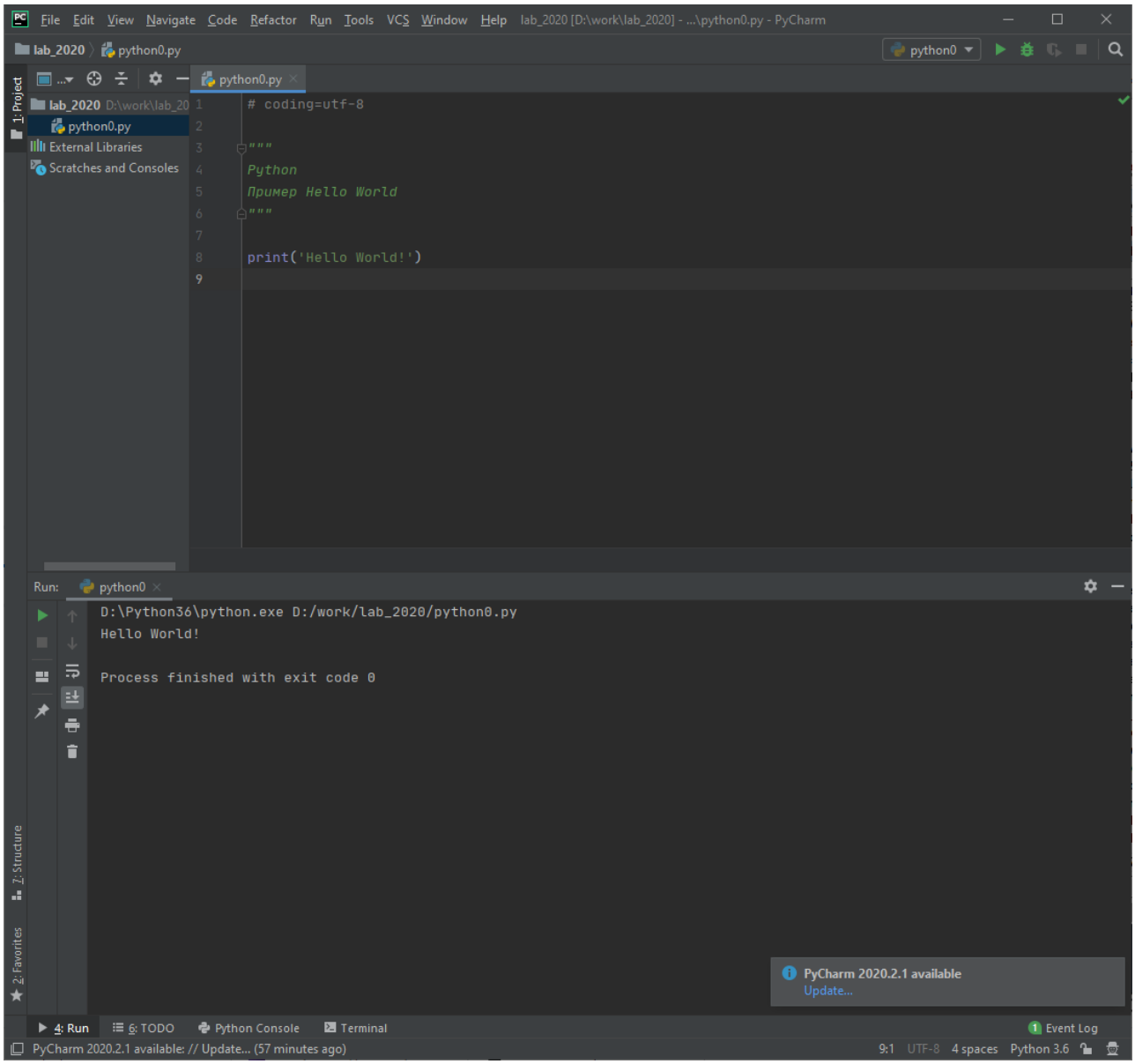
OpenGL

Python

Тема 0. Пример “Hello World”

Запустить пример “Hello World”:

```
# coding=utf-8


"""
Python
Пример Hello World
"""


print('Hello World!')
```



Тема 1. График функции

Варианты заданий:

3.1. $y = x^2 + \frac{16}{x} - 16, \quad [1, 4].$

3.2. $y = 4 - x - \frac{4}{x^2}, \quad [1, 4].$

3.3. $y = \sqrt[3]{2(x-2)^2(8-x)} - 1, \quad [0, 6].$

3.4. $y = \frac{2(x^2+3)}{x^2-2x+5}, \quad [-3, 3].$

3.5. $y = 2\sqrt{x} - x, \quad [0, 4].$

3.6. $y = 1 + \sqrt[3]{2(x-1)^2(x-7)}, \quad [-1, 5].$

3.7. $y = x - 4\sqrt{x} + 5, \quad [1, 9].$

3.8. $y = \frac{10x}{1+x^2}, \quad [0, 3].$

3.9. $y = \sqrt[3]{2(x+1)^2(5-x)} - 2, \quad [-3, 3].$

3.10. $y = 2x^2 + \frac{108}{x} - 59, \quad [2, 4].$

3.11. $y = 3 - x - \frac{4}{(x+2)^2}, \quad [-1, 2].$

3.12. $y = \sqrt[3]{2x^2(x-3)}, \quad [-1, 6].$

3.13. $y = \frac{2(-x^2+7x-7)}{x^2-2x+2}, \quad [1, 4].$

3.14. $y = x - 4\sqrt{x+2} + 8, \quad [-1, 7].$

3.15. $y = \sqrt[3]{2(x-2)^2(5-x)}, \quad [1, 5].$

3.16. $y = \frac{4x}{4+x^2}, \quad [-4, 2].$

3.17. $y = -\frac{x^2}{2} + \frac{8}{x} + 8, \quad [-4, -1].$

3.18. $y = \sqrt[3]{2x^2(x-6)}, \quad [-2, 4].$

3.19. $y = \frac{-2x(2x+3)}{x^2+4x+5}, \quad [1, 4].$

3.20. $y = -\frac{2(x^2+3)}{x^2+2x+5}, \quad [-5, 1].$

3.21. $y = \sqrt[3]{2(x-1)^2(x-4)}, \quad [0, 4].$

3.22. $y = x^2 - 2x + \frac{16}{x-1} - 13, \quad [2, 5].$

3.23. $y = 2\sqrt{x-1} - x + 2, \quad [1, 5].$

3.24. $y = \sqrt[3]{2(x+2)^2(1-x)}, \quad [-3, 4].$

3.25. $y = -\frac{x^2}{2} + 2x + \frac{8}{x-2} + 5, \quad [-2, 1].$

3.26. $y = 8x + \frac{4}{x^2} - 15, \quad \left[\frac{1}{2}, 2\right].$

3.27. $y = \sqrt[3]{2(x+2)^2(x-4)} + 3, \quad [-4, 2].$

3.28. $y = x^2 + 4x + \frac{16}{x+2} - 9, \quad [-1, 2].$

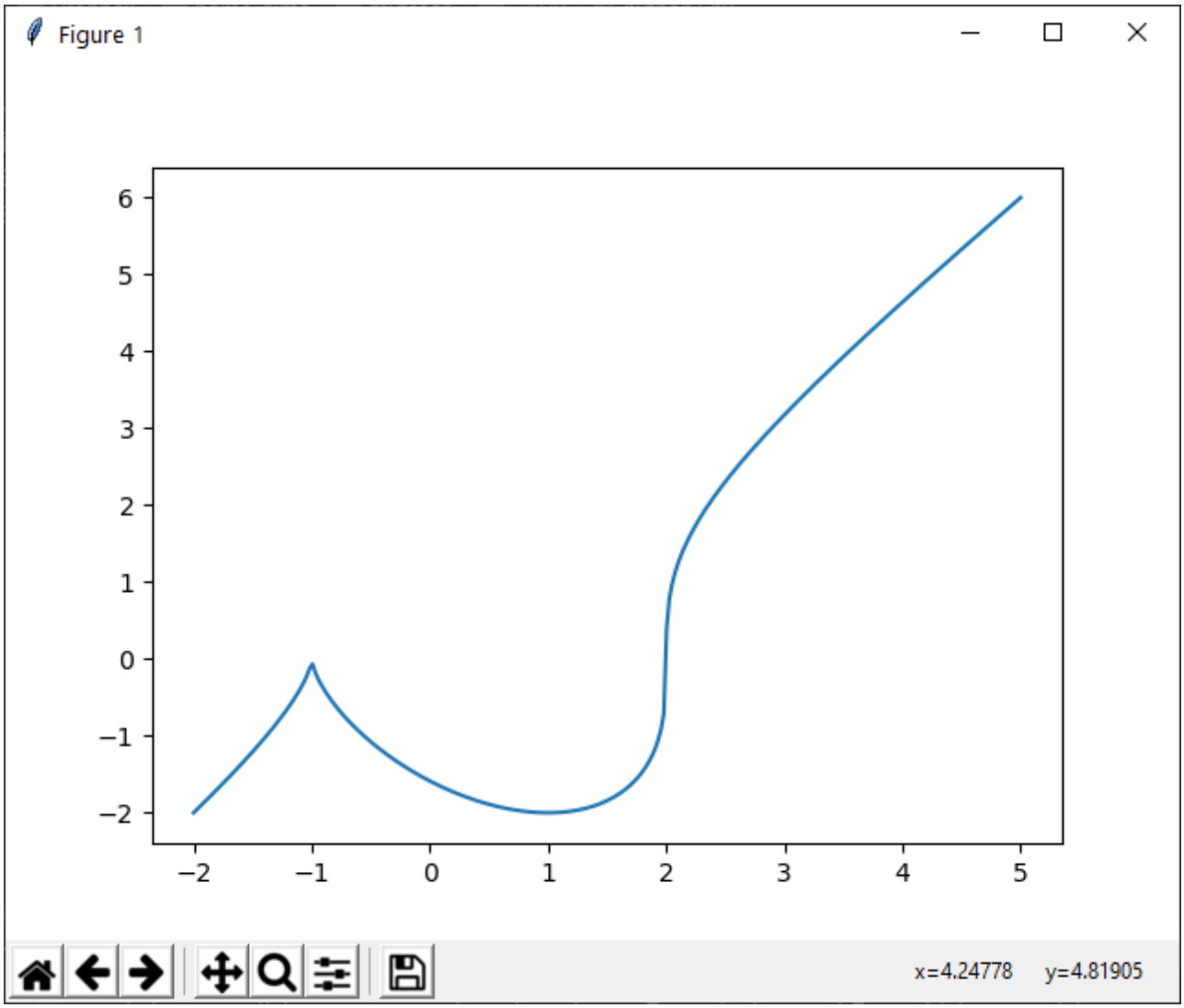
3.29. $y = \frac{4}{x^2} - 8x - 15, \quad \left[-2, -\frac{1}{2}\right].$

3.30. $y = \sqrt[3]{2(x+1)^2(x-2)}, \quad [-2, 5].$

3.31. $y = -\frac{10x+10}{x^2+2x+2}, \quad [-1, 2].$

Пример выполнения (вариант 3.30):

```
# coding=utf-8

"""
Python
График
"""

import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(-2, 5, 300)
y = 2 * (x + 1) ** 2 * (x - 2)
y = np.abs(y) ** (1 / 3) * np.sign(y)
# print(x, y)
plt.plot(x, y)
plt.show()
```



OpenGL

Варианты задания для тем 0 — 1

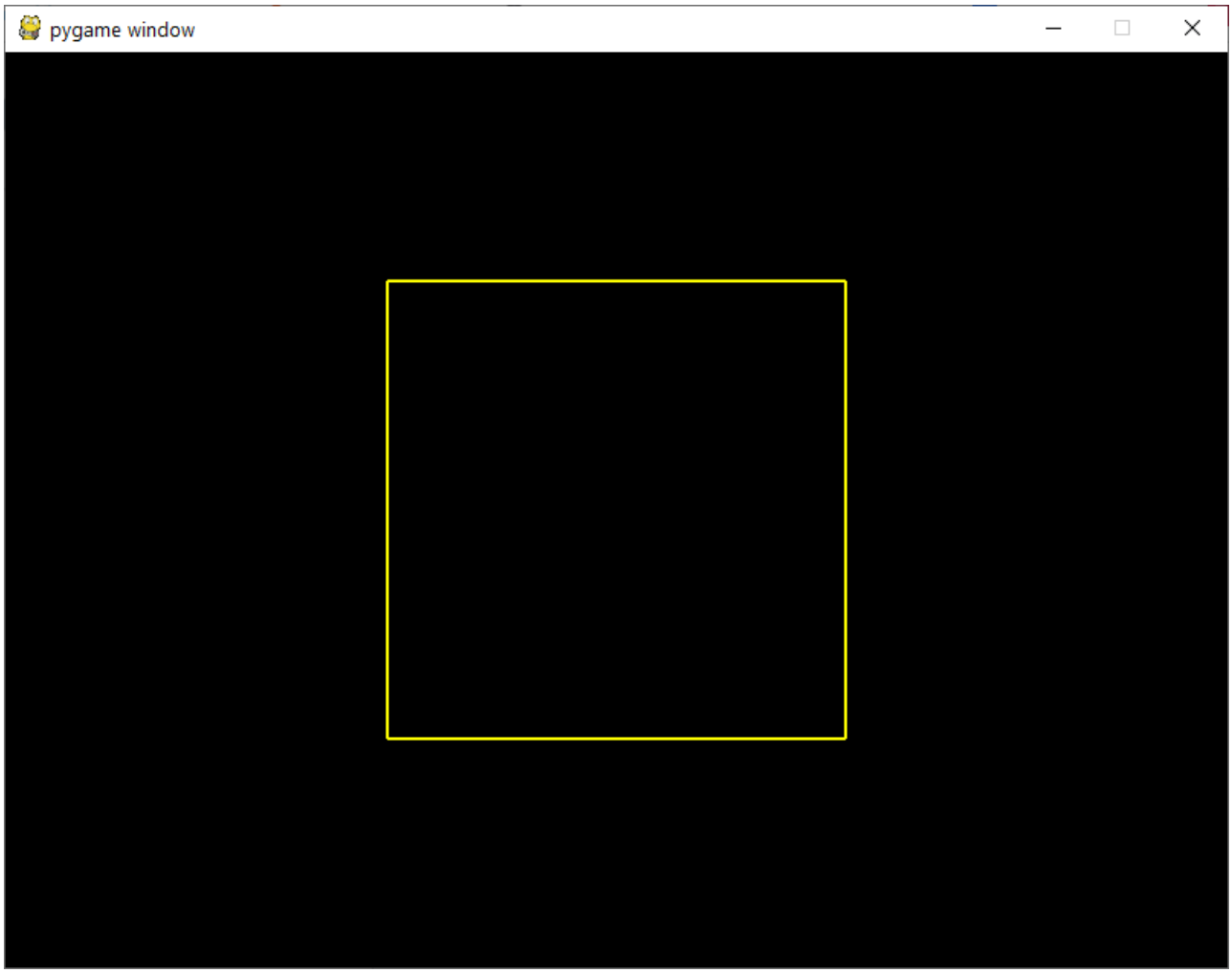
Вариант	Фигура	Оттенок цвета
1 — 10	Треугольник	Красный
11 — 20	Выпуклый четырехугольник	Зеленый
21 — 30	Пятиугольник	Синий
31 — 36	Невыпуклый четырехугольник	Маджента

Тема 0. Hello World, квадрат

Пример выполнения:

```
# coding=utf-8

"""
Hello World
Квадрат
"""

import pygame
from pygame.locals import *
from OpenGL.GL import *
from OpenGL.GLU import *

pygame.init()
size = (800, 600)
screen = pygame.display.set_mode(size, OPENGL | HWSURFACE | DO
UBLEBUF)

aspect = size[0] / size[1]
length = 2
glOrtho(-length * aspect, length * aspect, -length, length, -l
ength, length)

glEnable(GL_LINE_SMOOTH)

glClear(GL_COLOR_BUFFER_BIT)

glBegin(GL_LINES)
glColor3f(1, 1, 0)
glVertex3f(1, -1, 0)
glVertex3f(1, 1, 0)
```

```
glVertex3f(1, 1, 0)
glVertex3f(-1, 1, 0)
glVertex3f(-1, 1, 0)
glVertex3f(-1, -1, 0)
glVertex3f(-1, -1, 0)
glVertex3f(1, -1, 0)
glEnd()

pygame.display.flip()

is_loop = True
while is_loop:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            is_loop = False

pygame.quit()
```

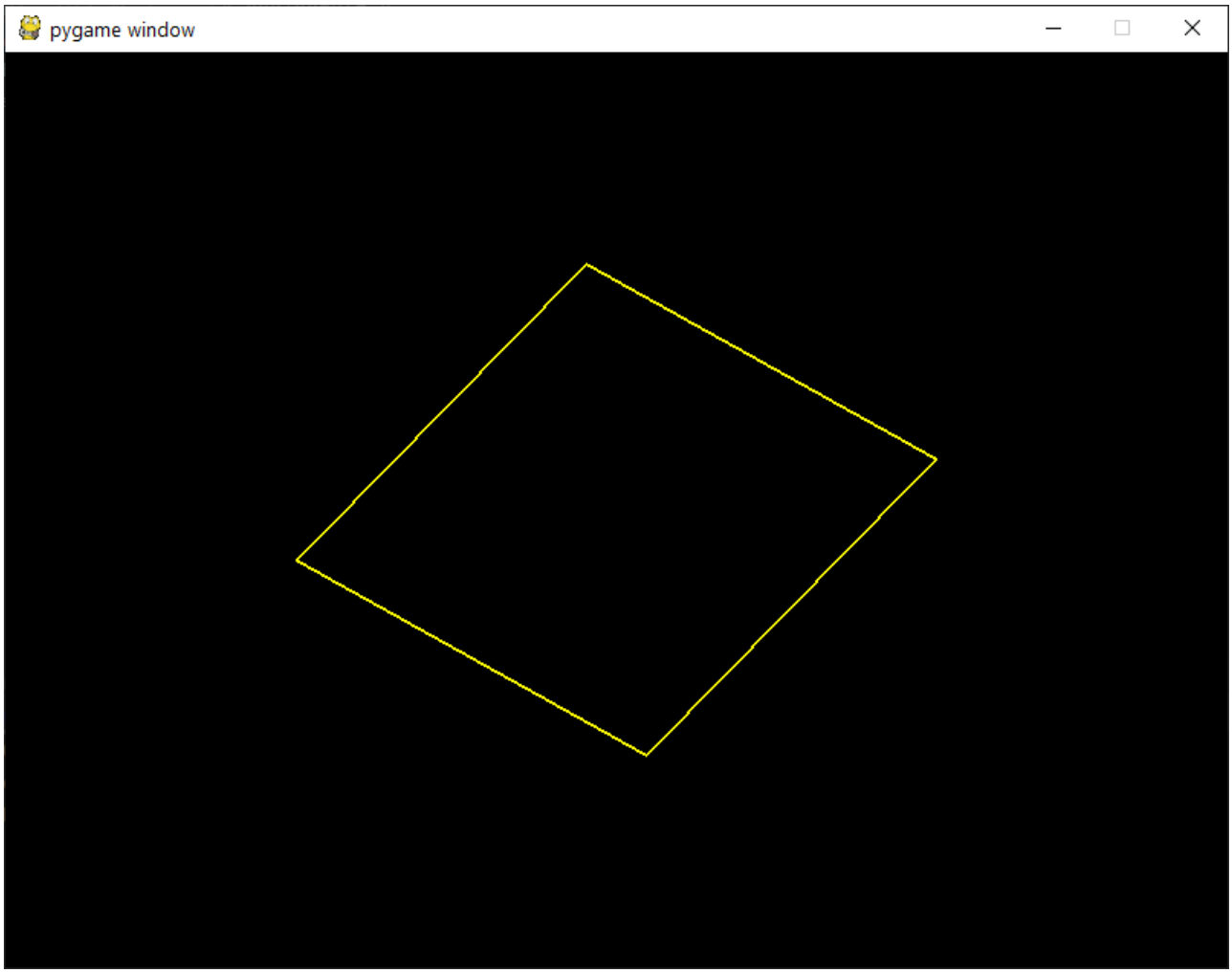


Тема 1. Квадрат, вращение

```
# coding=utf-8


"""
Квадрат
Вращение
"""


import pygame
from pygame.locals import *
from OpenGL.GL import *
```

```
from OpenGL.GLU import *



def draw():
    glBegin(GL_LINES)
    glColor3f(1, 1, 0)
    glVertex3f(1, -1, 0)
    glVertex3f(1, 1, 0)
    glVertex3f(1, 1, 0)
    glVertex3f(-1, 1, 0)
    glVertex3f(-1, 1, 0)
    glVertex3f(-1, -1, 0)
    glVertex3f(-1, -1, 0)
    glVertex3f(1, -1, 0)
    glEnd()



def main():
    pygame.init()
    size = (800, 600)
    screen = pygame.display.set_mode(size, OPENGL | HWSURFACE
| DOUBLEBUF)

    aspect = size[0] / size[1]
    length = 2
    glOrtho(-length * aspect, length * aspect, -length, lengt
h, -length, length)

    glEnable(GL_LINE_SMOOTH)
```

```
    clock = pygame.time.Clock()
    is_loop = True
    while is_loop:
        clock.tick(60)
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                is_loop = False
            if event.type == KEYUP and event.key == K_q:
                is_loop = False
        glClear(GL_COLOR_BUFFER_BIT)
        glRotatef(1, 1, 3, 1)
        draw()
        pygame.display.flip()

    pygame.quit()


main()
```

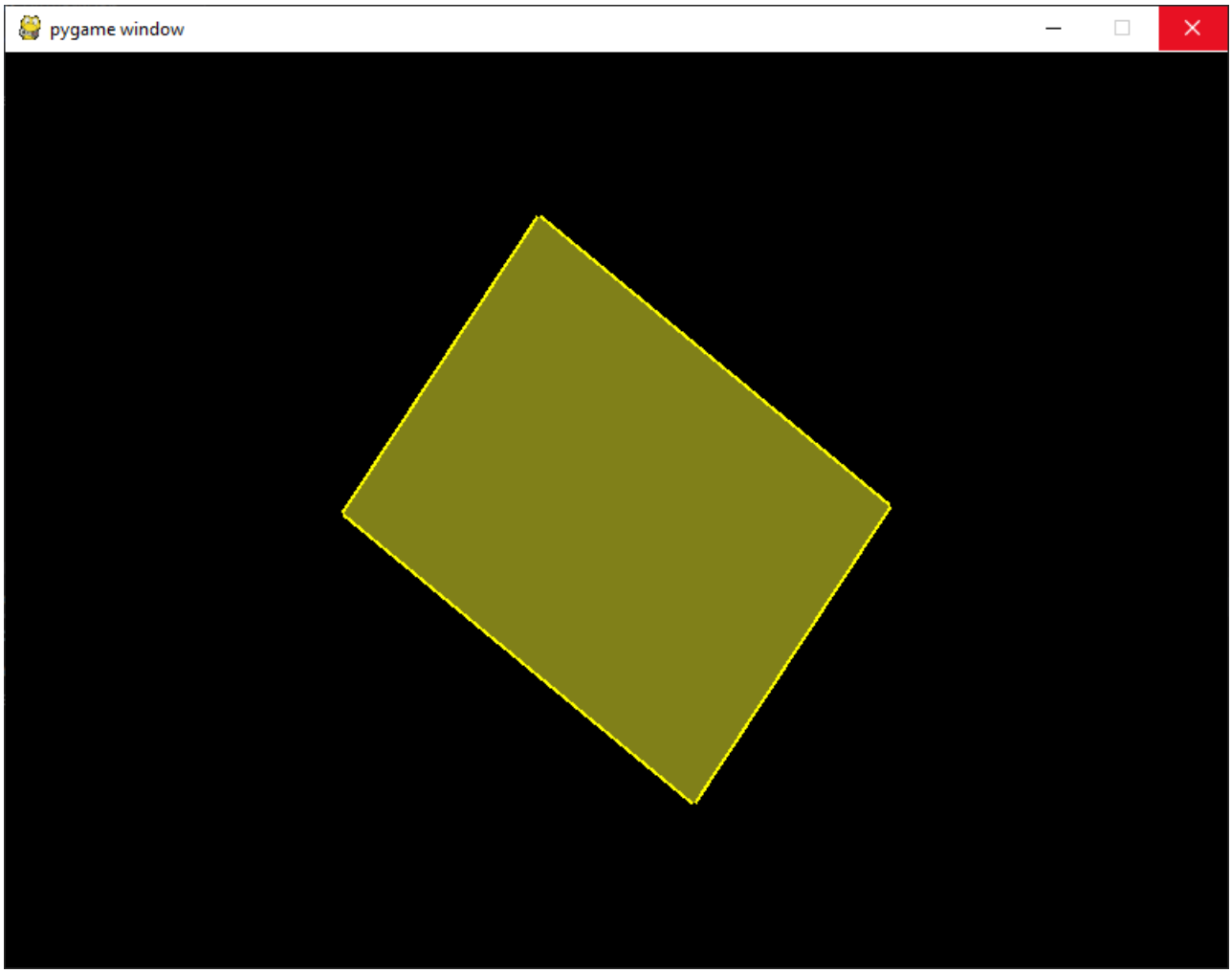


Тема 1.2. Квадрат, полигон

```
# coding=utf-8


"""
Квадрат, полигон
"""


import pygame
from pygame.locals import *
from OpenGL.GL import *
from OpenGL.GLU import *
```

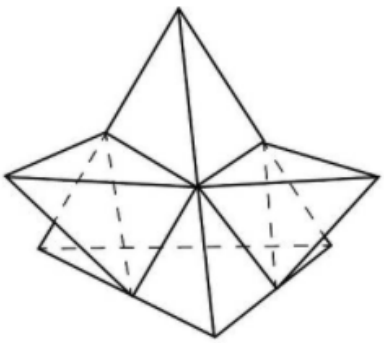
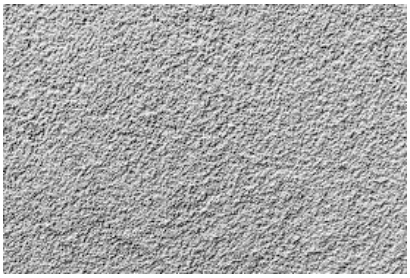
```
def draw():
    glLineWidth(3)
    glBegin(GL_LINES)
    glColor3f(1, 1, 0)
    glVertex3f(1, -1, 0)
    glVertex3f(1, 1, 0)
    glVertex3f(1, 1, 0)
    glVertex3f(-1, 1, 0)
    glVertex3f(-1, 1, 0)
    glVertex3f(-1, -1, 0)
    glVertex3f(-1, -1, 0)
    glVertex3f(1, -1, 0)
    glEnd()
    glLineWidth(1)
    glBegin(GL_QUADS)
    glColor3f(0.5, 0.5, 0.1)
    glVertex3f(-1, -1, 0)
    glVertex3f(1, -1, 0)
    glVertex3f(1, 1, 0)
    glVertex3f(-1, 1, 0)
    glEnd()


def main():
    pygame.init()
    size = (800, 600)
    screen = pygame.display.set_mode(size, OPENGL | HWSURFACE
| DOUBLEBUF)
```

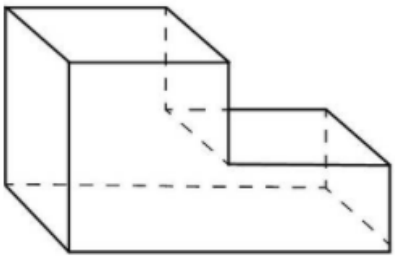
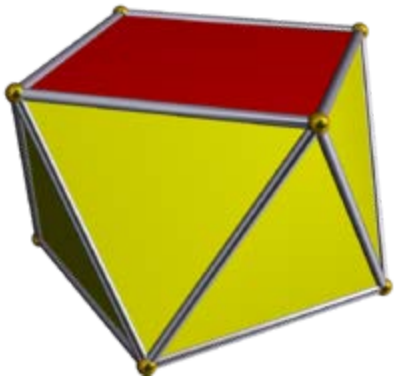
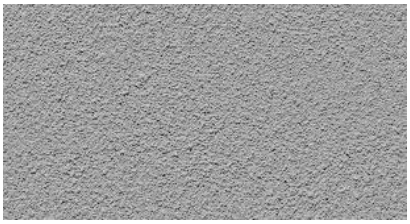
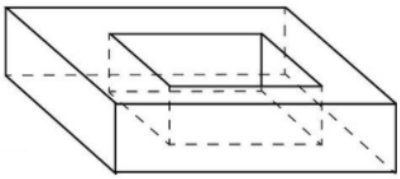
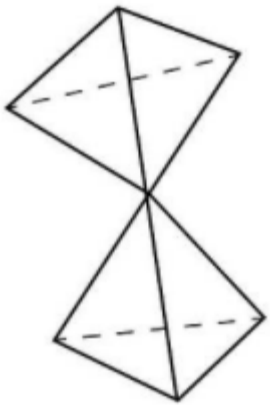
```
    aspect = size[0] / size[1]
    length = 2
    glOrtho(-length * aspect, length * aspect, -length, lengt
h, -length, length)

    glEnable(GL_LINE_SMOOTH)

    clock = pygame.time.Clock()
    is_loop = True
    while is_loop:
        clock.tick(60)
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                is_loop = False
            if event.type == KEYUP and event.key == K_q:
                is_loop = False
        glClear(GL_COLOR_BUFFER_BIT)
        glRotatef(1, 1, 3, 1)
        draw()
        pygame.display.flip()

    pygame.quit()


main()
```

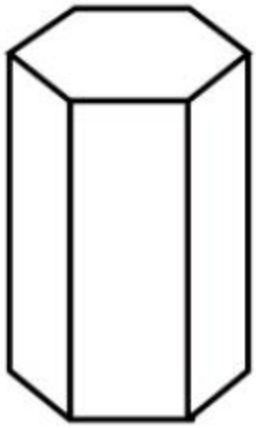
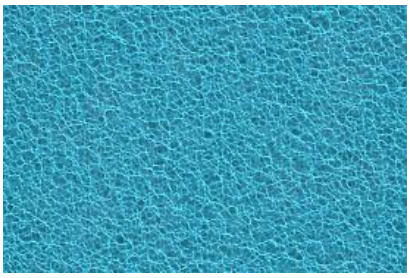
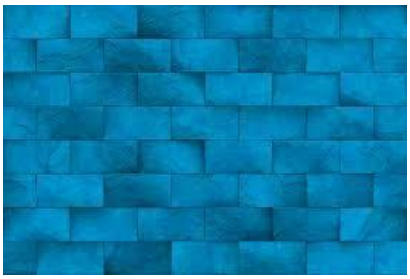
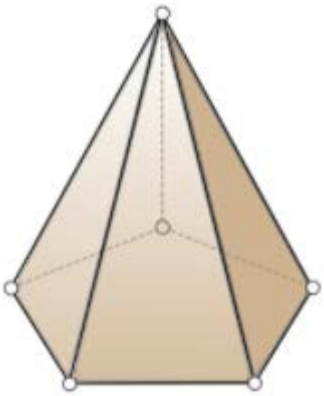
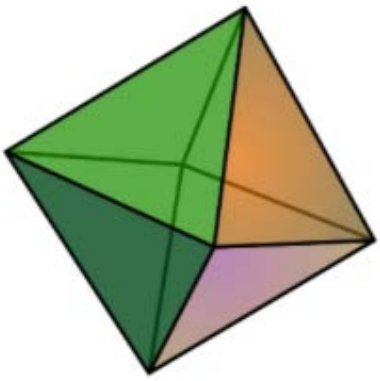


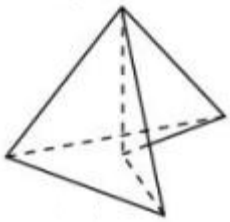
Варианты задания для тем 2 — 6

Создать сцену, содержащую вариацию заданного тела с требуемыми параметрами.

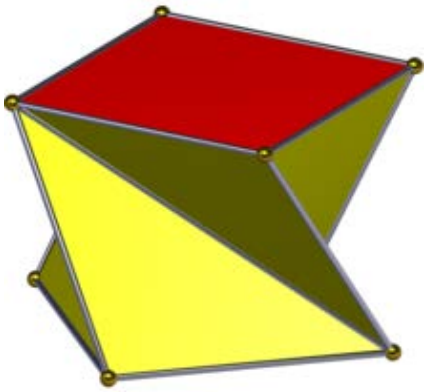
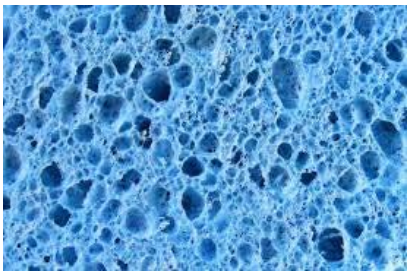
№	Тело	Цветовая схема	Текстура
1		viridis	

2		plasma	
3		inferno	
4		magma	
5		cividis	
6		Greys	

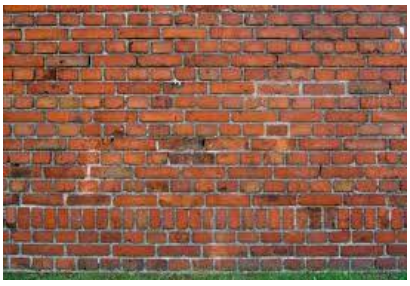
			
7		Purples	
8		Blues	
9		Greens	

10		Oranges	
11		Reds	
12		YlOrBr	
13		YlOrRd	
14		OrRd	
15		PuRd	

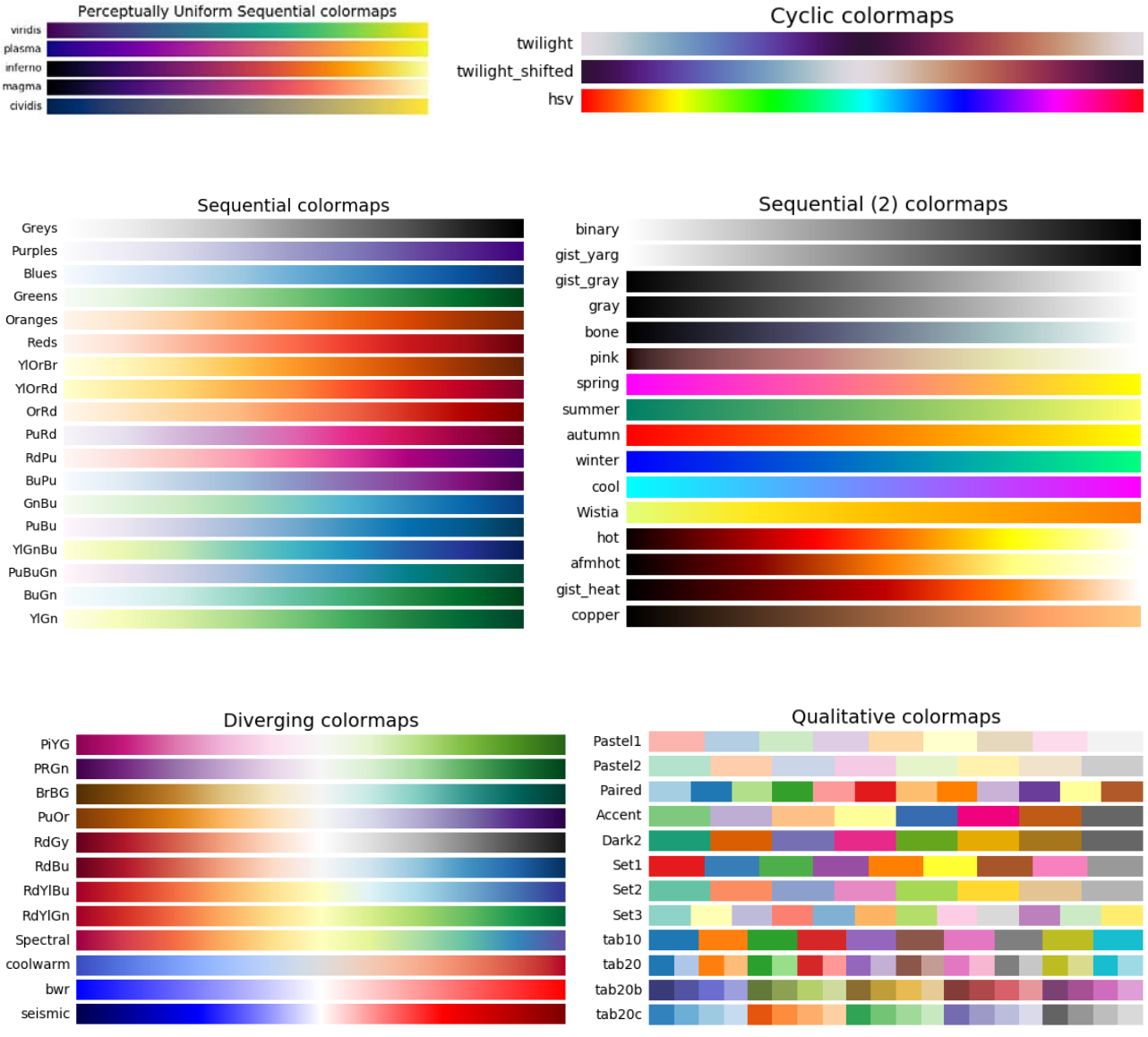
			
16		RdPu	
17		BuPu	
18		GnBu	
19		PuBu	
20		YlGnBu	

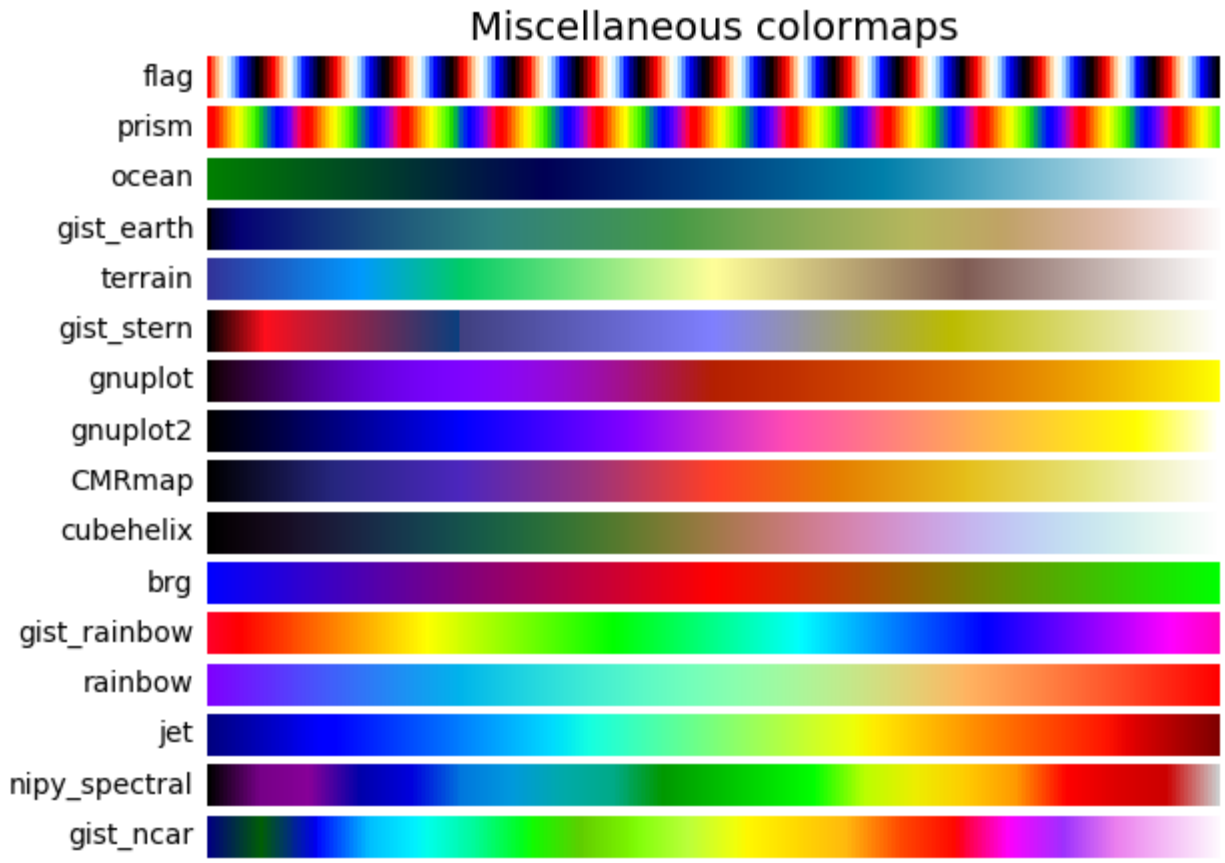
21		PuBuGn	
22	Карандаш	BuGn	
23	Табуретка	YlGn	
24	Стул	spring	
25	Стол	summer	
26	Раскрытая книга	autumn	

			
27	Шкаф	winter	
28	Линейка “Угольник”	ocean	
29	Колона	gist_earth	
30	Домик	gist_stern	
31		cubehelix	

			
32		brg	
33		gist_rainbow	
34		rainbow	
35		jet	
36		gist_ncar	

Примеры цветовых схем
(https://matplotlib.org/3.1.0/tutorials/colors/colormaps.html):





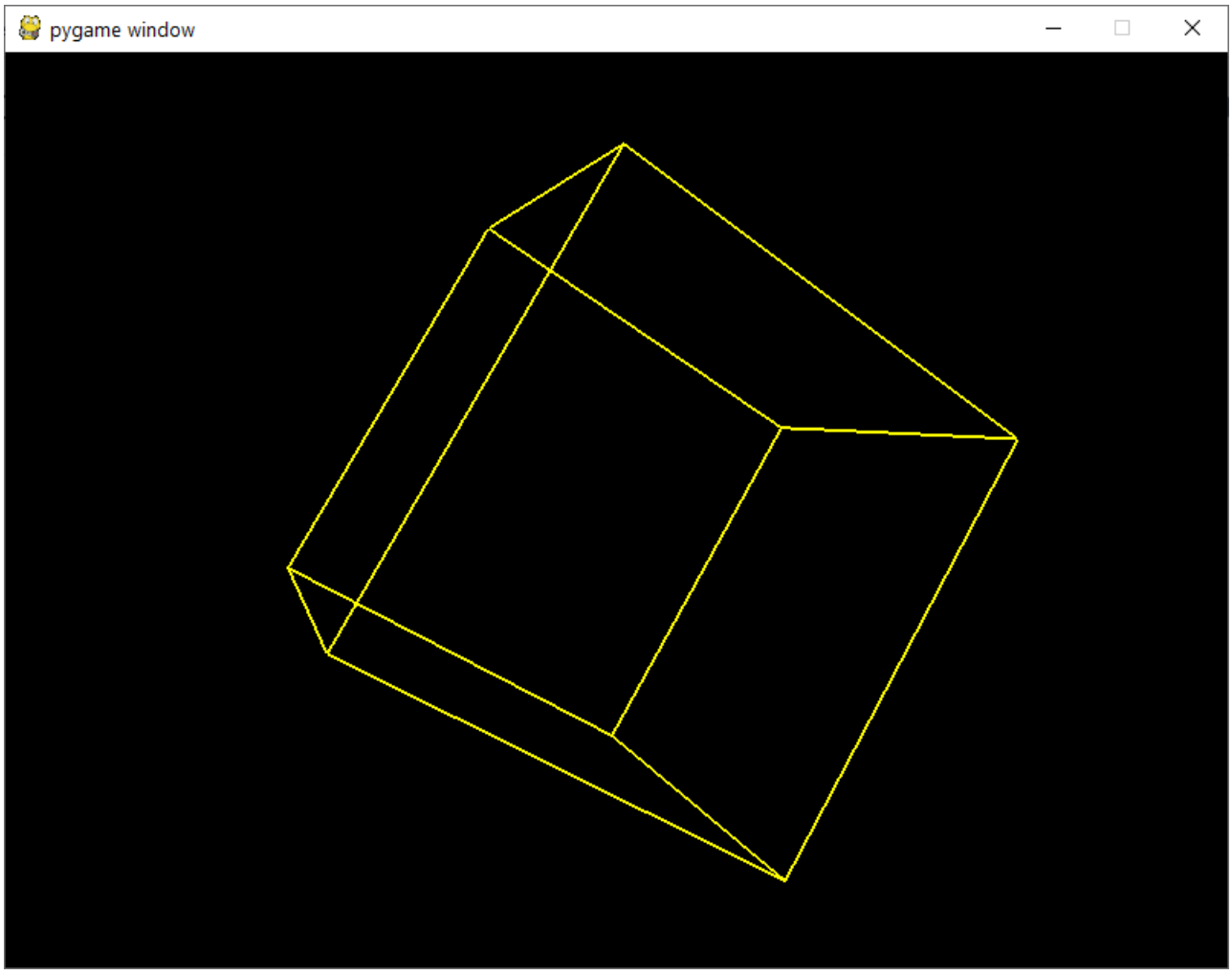
Тема 2. Куб, ребра

```
# coding=utf-8


"""
Куб, ребра
"""


import pygame
from pygame.locals import *
from OpenGL.GL import *
from OpenGL.GLU import *



def draw():
```

```
    vertices = [
        (1, -1, -1),
        (1, 1, -1),
        (-1, 1, -1),
        (-1, -1, -1),
        (1, -1, 1),
        (1, 1, 1),
        (-1, -1, 1),
        (-1, 1, 1),
    ]
    edges = [
        (0, 1),
        (0, 3),
        (0, 4),
        (2, 1),
        (2, 3),
        (2, 7),
        (6, 3),
        (6, 4),
        (6, 7),
        (5, 1),
        (5, 4),
        (5, 7),
    ]
    glBegin(GL_LINES)
    glColor3f(1, 1, 0)
    for edge in edges:
        glVertex3fv(vertices[edge[0]])
        glVertex3fv(vertices[edge[1]])
    glEnd()
```

```
def main():
    pygame.init()
    size = (800, 600)
    screen = pygame.display.set_mode(size, OPENGL | HWSURFACE
| DOUBLEBUF)

    aspect = size[0] / size[1]
    # length = 2
    # glOrtho(-length * aspect, length * aspect, -length, leng
th, -length, length)
    gluPerspective(45, aspect, 0.1, 50)
    glTranslatef(0, 0, -5)

    glEnable(GL_LINE_SMOOTH)

    clock = pygame.time.Clock()
    is_loop = True
    while is_loop:
        clock.tick(60)
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                is_loop = False
            if event.type == KEYUP and event.key == K_q:
                is_loop = False
        glClear(GL_COLOR_BUFFER_BIT)
        glRotatef(1, 1, 3, 1)
        draw()
        pygame.display.flip()
```

```
    pygame.quit()


main()
```



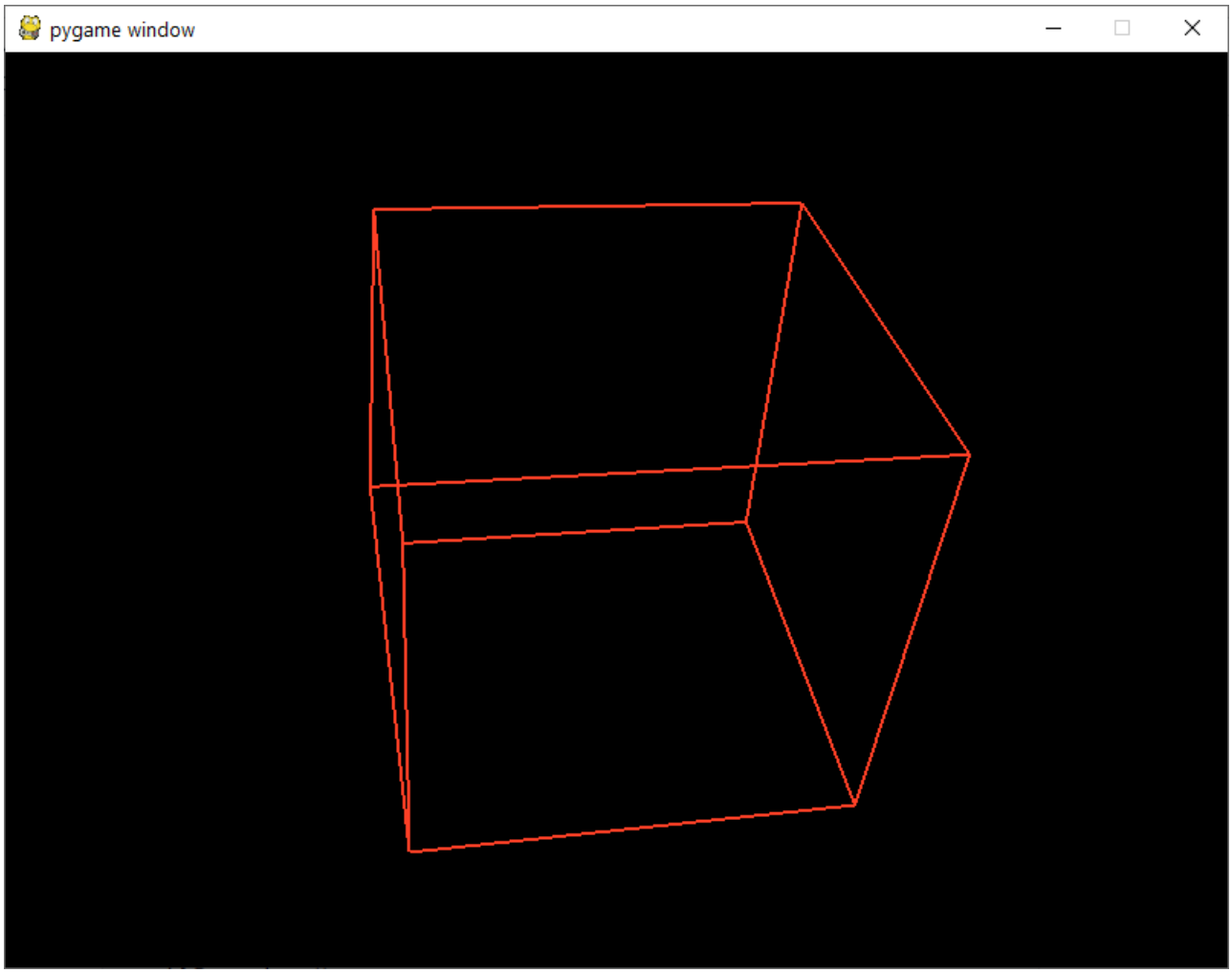
Тема 2.2. Куб, ребра, цветовая схема

```
# coding=utf-8


"""
Куб, ребра, цветовая схема
```

```
"""

import pygame
from pygame.locals import *
from OpenGL.GL import *
from OpenGL.GLU import *

import numpy as np
import matplotlib.cm


def get_colors(cname, cnum=1):
    cnum_max = cnum + 2
    cmap = matplotlib.cm.get_cmap(cname, cnum_max)
    colors = np.array(cmap(np.linspace(1, 0, cnum_max)))
    colors = colors[1:cnum + 1, :3]
    return np.squeeze(colors)


def draw():
    vertices = [
        (1, -1, -1),
        (1, 1, -1),
        (-1, 1, -1),
        (-1, -1, -1),
        (1, -1, 1),
        (1, 1, 1),
        (-1, -1, 1),
        (-1, 1, 1),
    ]
```

```
    edges = [
        (0, 1),
        (0, 3),
        (0, 4),
        (2, 1),
        (2, 3),
        (2, 7),
        (6, 3),
        (6, 4),
        (6, 7),
        (5, 1),
        (5, 4),
        (5, 7),
    ]
    glBegin(GL_LINES)
    # glColor3f(1, 1, 0)
    color = get_colors('CMRmap')
    glColor3f(*color)
    for edge in edges:
        glVertex3fv(vertices[edge[0]])
        glVertex3fv(vertices[edge[1]])
    glEnd()


def main():
    pygame.init()
    size = (800, 600)
    screen = pygame.display.set_mode(size, OPENGL | HWSURFACE
| DOUBLEBUF)
```

```
    aspect = size[0] / size[1]
    # length = 2
    # glOrtho(-length * aspect, length * aspect, -length, leng
th, -length, length)
    gluPerspective(45, aspect, 0.1, 50)
    glTranslatef(0, 0, -5)

    glEnable(GL_LINE_SMOOTH)

    clock = pygame.time.Clock()
    is_loop = True
    while is_loop:
        clock.tick(60)
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                is_loop = False
            if event.type == KEYUP and event.key == K_q:
                is_loop = False
        glClear(GL_COLOR_BUFFER_BIT)
        glRotatef(1, 1, 3, 1)
        draw()
        pygame.display.flip()

    pygame.quit()



main()
```



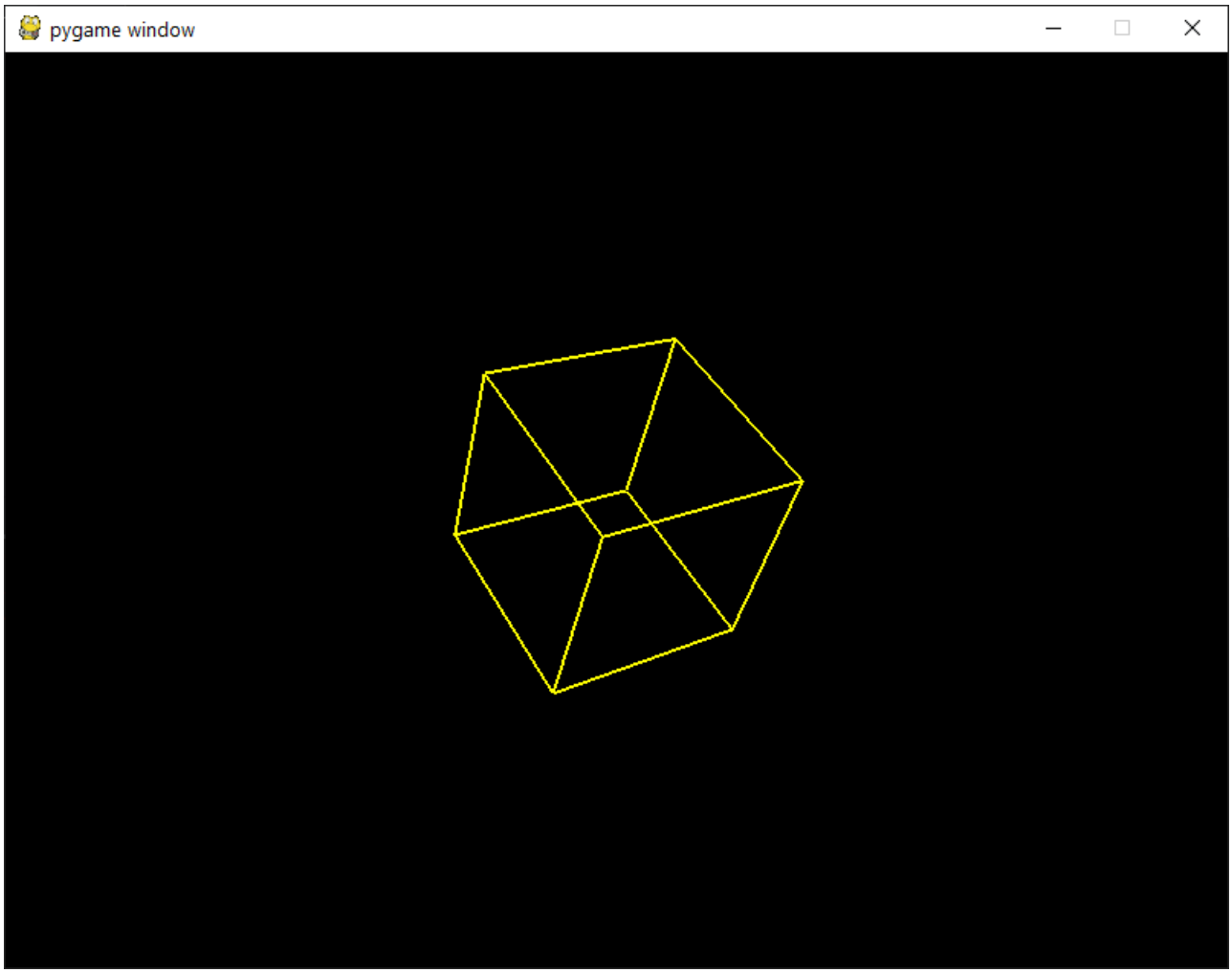
Тема 3. Куб, ребра, класс

```
# coding=utf-8


"""
Куб, ребра, класс
"""


import pygame
from pygame.locals import *
from OpenGL.GL import *
from OpenGL.GLU import *
```

```
class Cube:
    def __init__(self):
        self.vertices = [
            (1, -1, -1),
            (1, 1, -1),
            (-1, 1, -1),
            (-1, -1, -1),
            (1, -1, 1),
            (1, 1, 1),
            (-1, -1, 1),
            (-1, 1, 1),
        ]
        self.edges = [
            (0, 1),
            (0, 3),
            (0, 4),
            (2, 1),
            (2, 3),
            (2, 7),
            (6, 3),
            (6, 4),
            (6, 7),
            (5, 1),
            (5, 4),
            (5, 7),
        ]

    def draw(self):
        glBegin(GL_LINES)
```

```
        glColor3f(1, 1, 0)
        for edge in self.edges:
            glVertex3fv(self.vertices[edge[0]])
            glVertex3fv(self.vertices[edge[1]])
        glEnd()


def main():
    pygame.init()
    size = (800, 600)
    screen = pygame.display.set_mode(size, OPENGL | HWSURFACE
| DOUBLEBUF)

    aspect = size[0] / size[1]
    gluPerspective(45, aspect, 0.1, 50)
    glTranslatef(0, 0, -10)

    glEnable(GL_LINE_SMOOTH)

    cube = Cube()

    clock = pygame.time.Clock()
    is_rotate = True
    is_loop = True
    while is_loop:
        clock.tick(60)
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                is_loop = False
            if event.type == KEYUP and event.key == K_q:
```

```
                is_loop = False
            if event.type == KEYUP and event.key == K_SPACE:
                is_rotate = not is_rotate
        glClear(GL_COLOR_BUFFER_BIT)
        if is_rotate:
            glRotatef(1, 1, 3, 1)
        cube.draw()
        pygame.display.flip()

    pygame.quit()


main()
```



Тема 4. Куб, полигоны

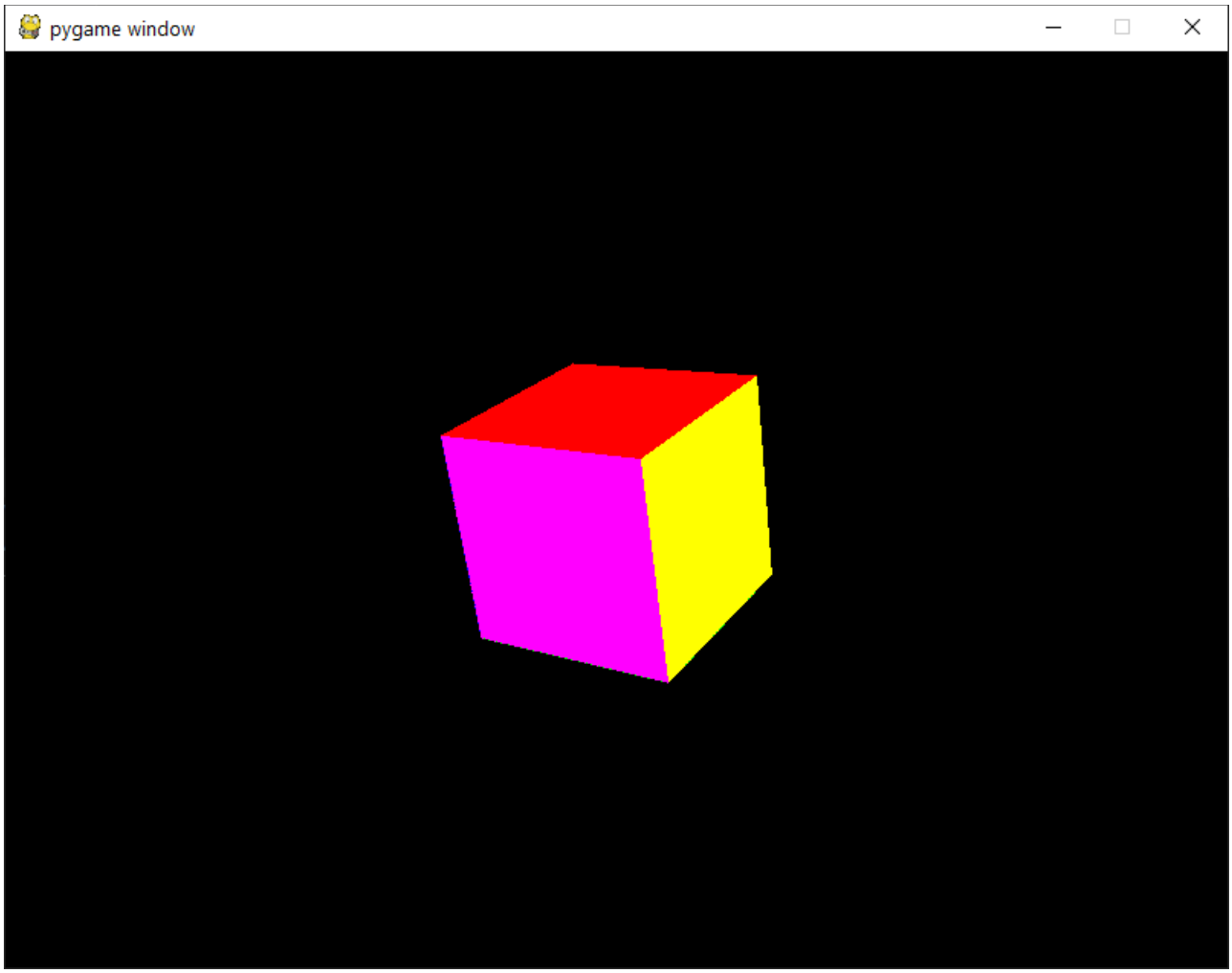
```
# coding=utf-8


"""
Куб, полигоны
"""


import pygame
from pygame.locals import *
from OpenGL.GL import *
from OpenGL.GLU import *
```

```
class Cube:
    def __init__(self):
        self.vertices = [
            (1, -1, -1),
            (1, 1, -1),
            (-1, 1, -1),
            (-1, -1, -1),
            (1, -1, 1),
            (1, 1, 1),
            (-1, -1, 1),
            (-1, 1, 1),
        ]
        self.edges = [
            (0, 1),
            (0, 3),
            (0, 4),
            (2, 1),
            (2, 3),
            (2, 7),
            (6, 3),
            (6, 4),
            (6, 7),
            (5, 1),
            (5, 4),
            (5, 7),
        ]
        self.faces = [
            (1, 2, 7, 5),
            (4, 6, 3, 0),
```

```
            (5, 7, 6, 4),
            (0, 3, 2, 1),
            (7, 2, 3, 6),
            (1, 5, 4, 0),
        ]
        self.colors = [
            (1, 0, 0),
            (0, 1, 0),
            (1, 1, 0),
            (0, 0, 1),
            (1, 0, 1),
            (0, 1, 1),
        ]

    def draw_edges(self):
        glBegin(GL_LINES)
        glColor3f(1, 1, 0)
        for edge in self.edges:
            glVertex3fv(self.vertices[edge[0]])
            glVertex3fv(self.vertices[edge[1]])
        glEnd()

    def draw_polygons(self):
        glBegin(GL_QUADS)
        for fi, face in enumerate(self.faces):
            glColor3fv(self.colors[fi])
            for vi in face:
                glVertex3fv(self.vertices[vi])
        glEnd()
```

```
    def draw(self):
        # self.draw_edges()
        self.draw_polygons()



def main():
    pygame.init()
    size = (800, 600)
    screen = pygame.display.set_mode(size, OPENGL | HWSURFACE
| DOUBLEBUF)

    aspect = size[0] / size[1]
    gluPerspective(45, aspect, 0.1, 50)
    glTranslatef(0, 0, -10)

    glEnable(GL_LINE_SMOOTH)
    glEnable(GL_MULTISAMPLE)
    glEnable(GL_DEPTH_TEST)

    cube = Cube()

    clock = pygame.time.Clock()
    is_rotate = True
    is_loop = True
    while is_loop:
        clock.tick(60)
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                is_loop = False
            if event.type == KEYUP and event.key == K_q:
```

```
                is_loop = False
            if event.type == KEYUP and event.key == K_SPACE:
                is_rotate = not is_rotate
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
        if is_rotate:
            glRotatef(1, 1, 3, 1)
        cube.draw()
        pygame.display.flip()

    pygame.quit()


main()
```



Тема 4.2. Куб, полигоны, цветовая схема

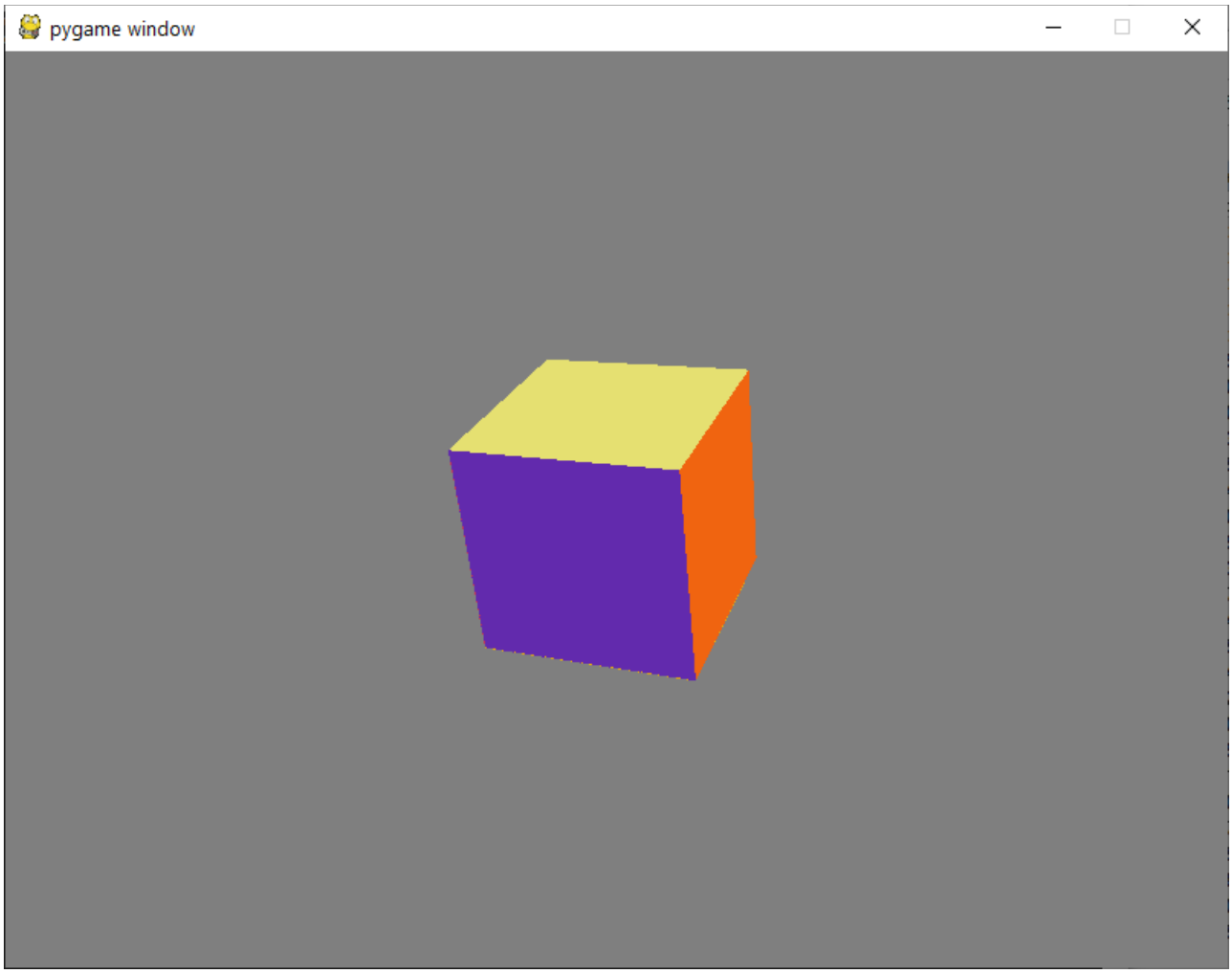
```
# coding=utf-8


"""
Куб, полигоны, цветовая схема
"""


import pygame
from pygame.locals import *
from OpenGL.GL import *
from OpenGL.GLU import *
```

```
import numpy as np
import matplotlib.cm


def get_colors(cname, cnum=1):
    cnum_max = cnum + 2
    cmap = matplotlib.cm.get_cmap(cname, cnum_max)
    colors = np.array(cmap(np.linspace(1, 0, cnum_max)))
    colors = colors[1:cnum + 1, :3]
    return np.squeeze(colors)


class Cube:
    def __init__(self):
        self.vertices = [
            (1, -1, -1),
            (1, 1, -1),
            (-1, 1, -1),
            (-1, -1, -1),
            (1, -1, 1),
            (1, 1, 1),
            (-1, -1, 1),
            (-1, 1, 1),
        ]
        self.edges = [
            (0, 1),
            (0, 3),
            (0, 4),
            (2, 1),
            (2, 3),
```

```
            (2, 7),
            (6, 3),
            (6, 4),
            (6, 7),
            (5, 1),
            (5, 4),
            (5, 7),
        ]
        self.faces = [
            (1, 2, 7, 5),
            (4, 6, 3, 0),
            (5, 7, 6, 4),
            (0, 3, 2, 1),
            (7, 2, 3, 6),
            (1, 5, 4, 0),
        ]
        # self.colors = [
        #     (1, 0, 0),
        #     (0, 1, 0),
        #     (1, 1, 0),
        #     (0, 0, 1),
        #     (1, 0, 1),
        #     (0, 1, 1),
        # ]
        self.colors = get_colors('CMRmap', 6)

    def draw_edges(self):
        glBegin(GL_LINES)
        glColor3f(1, 1, 0)
        for edge in self.edges:
```

```
            glVertex3fv(self.vertices[edge[0]])
            glVertex3fv(self.vertices[edge[1]])
        glEnd()

    def draw_polygons(self):
        glBegin(GL_QUADS)
        for fi, face in enumerate(self.faces):
            glColor3fv(self.colors[fi])
            for vi in face:
                glVertex3fv(self.vertices[vi])
        glEnd()

    def draw(self):
        # self.draw_edges()
        self.draw_polygons()


def main():
    pygame.init()
    size = (800, 600)
    screen = pygame.display.set_mode(size, OPENGL | HWSURFACE
| DOUBLEBUF)

    aspect = size[0] / size[1]
    gluPerspective(45, aspect, 0.1, 50)
    glTranslatef(0, 0, -10)

    glEnable(GL_LINE_SMOOTH)
    glEnable(GL_MULTISAMPLE)
    glEnable(GL_DEPTH_TEST)
```

```
    glClearColor(0.5, 0.5, 0.5, 1)

    cube = Cube()

    clock = pygame.time.Clock()
    is_rotate = True
    is_loop = True
    while is_loop:
        clock.tick(60)
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                is_loop = False
            if event.type == KEYUP and event.key == K_q:
                is_loop = False
            if event.type == KEYUP and event.key == K_SPACE:
                is_rotate = not is_rotate
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
        if is_rotate:
            glRotatef(1, 1, 3, 1)
        cube.draw()
        pygame.display.flip()

    pygame.quit()


main()
```



Тема 5. Куб, трансформация

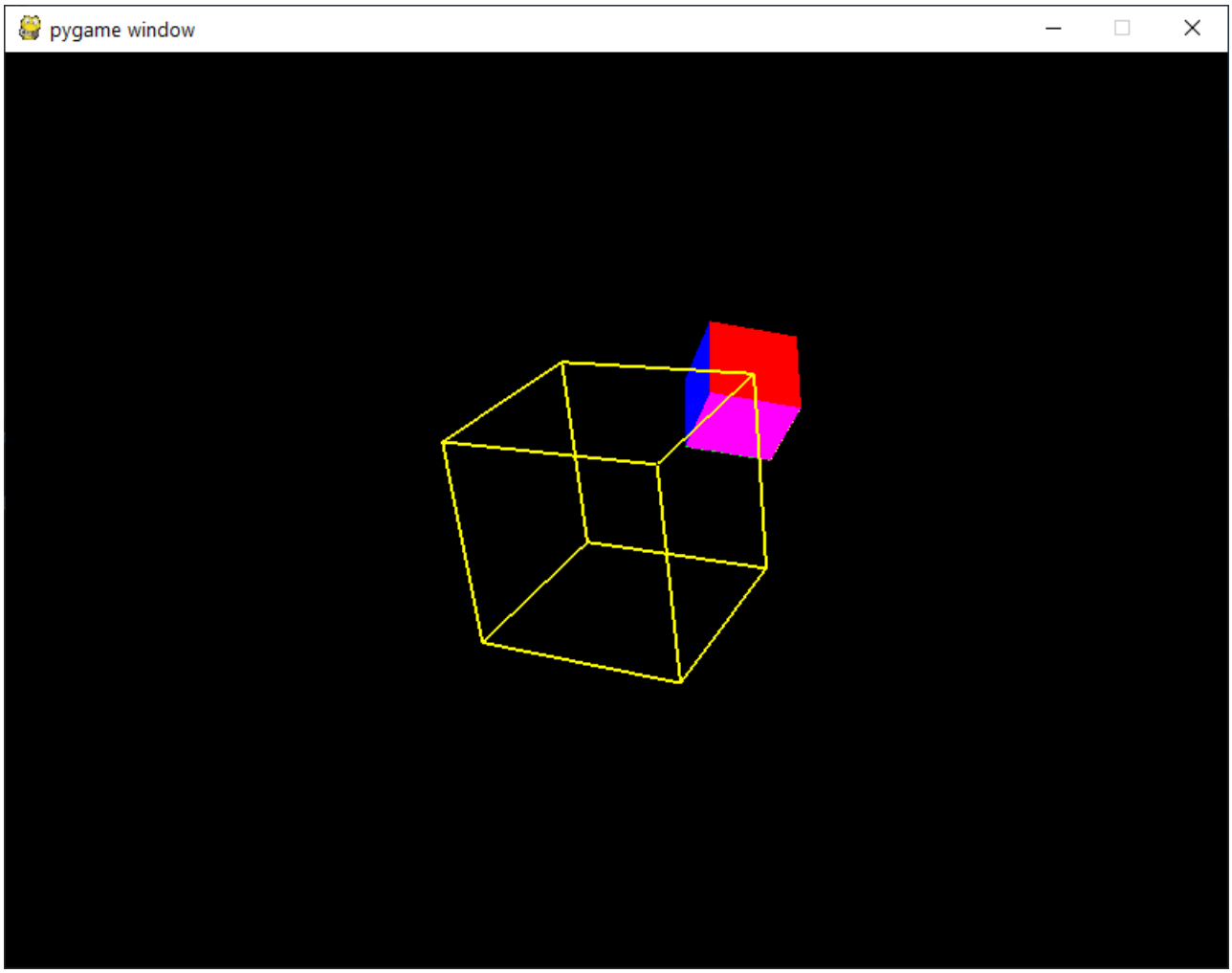
```
# coding=utf-8


"""
Куб, трансформация
"""


import pygame
from pygame.locals import *
from OpenGL.GL import *
from OpenGL.GLU import *
```

```
class Cube:
    def __init__(self):
        self.vertices = [
            (1, -1, -1),
            (1, 1, -1),
            (-1, 1, -1),
            (-1, -1, -1),
            (1, -1, 1),
            (1, 1, 1),
            (-1, -1, 1),
            (-1, 1, 1),
        ]
        self.edges = [
            (0, 1),
            (0, 3),
            (0, 4),
            (2, 1),
            (2, 3),
            (2, 7),
            (6, 3),
            (6, 4),
            (6, 7),
            (5, 1),
            (5, 4),
            (5, 7),
        ]
        self.faces = [
            (1, 2, 7, 5),
            (4, 6, 3, 0),
```

```
            (5, 7, 6, 4),
            (0, 3, 2, 1),
            (7, 2, 3, 6),
            (1, 5, 4, 0),
        ]
        self.colors = [
            (1, 0, 0),
            (0, 1, 0),
            (1, 1, 0),
            (0, 0, 1),
            (1, 0, 1),
            (0, 1, 1),
        ]

    def draw_edges(self):
        glBegin(GL_LINES)
        glColor3f(1, 1, 0)
        for edge in self.edges:
            glVertex3fv(self.vertices[edge[0]])
            glVertex3fv(self.vertices[edge[1]])
        glEnd()

    def draw_polygons(self):
        glBegin(GL_QUADS)
        for fi, face in enumerate(self.faces):
            glColor3fv(self.colors[fi])
            for vi in face:
                glVertex3fv(self.vertices[vi])
        glEnd()
```

```
    def draw(self):
        self.draw_edges()
        glPushMatrix()
        glTranslatef(3, 0, 0)
        glRotatef(45, 1, 1, 1)
        glScalef(0.5, 0.5, 0.5)
        self.draw_polygons()
        glPopMatrix()



def main():
    pygame.init()
    size = (800, 600)
    screen = pygame.display.set_mode(size, OPENGL | HWSURFACE
| DOUBLEBUF)

    aspect = size[0] / size[1]
    gluPerspective(45, aspect, 0.1, 50)
    glTranslatef(0, 0, -10)

    glEnable(GL_LINE_SMOOTH)
    glEnable(GL_MULTISAMPLE)
    glEnable(GL_DEPTH_TEST)

    cube = Cube()

    clock = pygame.time.Clock()
    is_rotate = True
    is_loop = True
    while is_loop:
```

```
        clock.tick(60)
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                is_loop = False
            if event.type == KEYUP and event.key == K_q:
                is_loop = False
            if event.type == KEYUP and event.key == K_SPACE:
                is_rotate = not is_rotate
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
        if is_rotate:
            glRotatef(1, 1, 3, 1)
        cube.draw()
        pygame.display.flip()

    pygame.quit()


main()
```



Тема 6. Куб, текстура

```
# coding=utf-8


"""
Куб, текстура
"""


import pygame
from pygame.locals import *
from OpenGL.GL import *
from OpenGL.GLU import *
from PIL.Image import open
```

```
def load_texture(filename):
    img = open(filename)
    img_data = img.tobytes('raw', 'RGB', 0, -1)
    texture_id = glGenTextures(1)
    glBindTexture(GL_TEXTURE_2D, texture_id)
    glPixelStorei(GL_UNPACK_ALIGNMENT, 1)
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAM
P)
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAM
P)
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_L
INEAR)
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_L
INEAR_MIPMAP_LINEAR)
    gluBuild2DMipmaps(GL_TEXTURE_2D, GL_RGB, img.size[0], img.
size[1], GL_RGB, GL_UNSIGNED_BYTE, img_data)
    return texture_id



class Cube:
    def __init__(self, texture_id=None):
        self.vertices = [
            (1, -1, -1),
            (1, 1, -1),
            (-1, 1, -1),
            (-1, -1, -1),
            (1, -1, 1),
            (1, 1, 1),
            (-1, -1, 1),
```

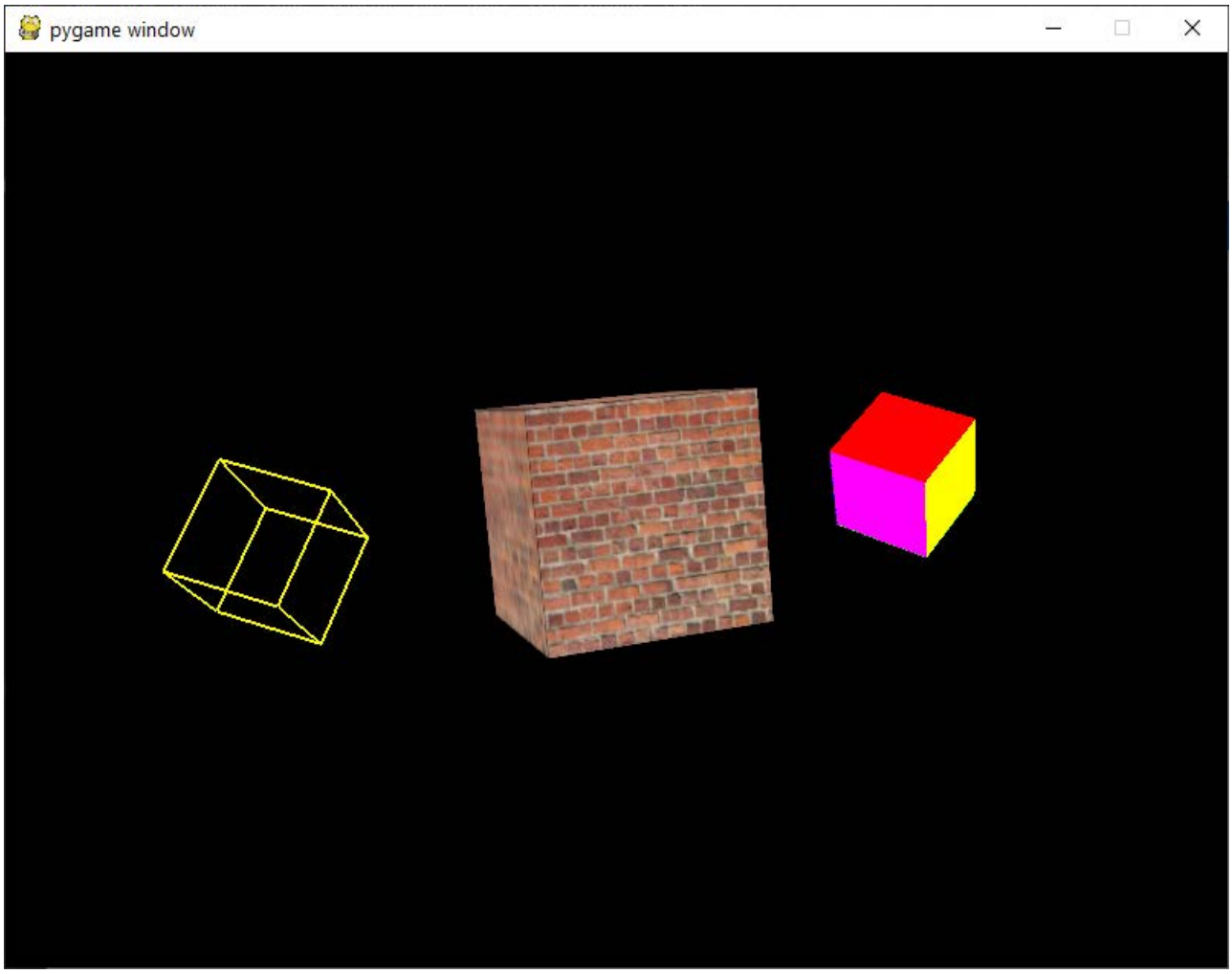
```
            (-1, 1, 1),
        ]
        self.edges = [
            (0, 1),
            (0, 3),
            (0, 4),
            (2, 1),
            (2, 3),
            (2, 7),
            (6, 3),
            (6, 4),
            (6, 7),
            (5, 1),
            (5, 4),
            (5, 7),
        ]
        self.faces = [
            (1, 2, 7, 5),
            (4, 6, 3, 0),
            (5, 7, 6, 4),
            (0, 3, 2, 1),
            (7, 2, 3, 6),
            (1, 5, 4, 0),
        ]
        self.colors = [
            (1, 0, 0),
            (0, 1, 0),
            (1, 1, 0),
            (0, 0, 1),
            (1, 0, 1),
```

```
            (0, 1, 1),
        ]
        self.uvs = [
            [(0, 0), (1, 0), (1, 1), (0, 1)],
            [(0, 0), (1, 0), (1, 1), (0, 1)],
            [(0, 0), (1, 0), (1, 1), (0, 1)],
            [(0, 0), (1, 0), (1, 1), (0, 1)],
            [(0, 0), (1, 0), (1, 1), (0, 1)],
            [(0, 0), (1, 0), (1, 1), (0, 1)],
        ]
        self.texture_id = texture_id

    def draw_edges(self):
        glBegin(GL_LINES)
        glColor3f(1, 1, 0)
        for edge in self.edges:
            glVertex3fv(self.vertices[edge[0]])
            glVertex3fv(self.vertices[edge[1]])
        glEnd()

    def draw_polygons(self):
        glBegin(GL_QUADS)
        for fi, face in enumerate(self.faces):
            glColor3fv(self.colors[fi])
            for vi in face:
                glVertex3fv(self.vertices[vi])
        glEnd()

    def draw_texture(self):
        if self.texture_id is None:
```

```
            return
        glEnable(GL_TEXTURE_2D)
        glBindTexture(GL_TEXTURE_2D, self.texture_id)
        glBegin(GL_QUADS)
        glColor3f(1, 1, 1)
        for fi, face in enumerate(self.faces):
            for i, vi in enumerate(face):
                glTexCoord2fv(self.uvs[fi][i])
                glVertex3fv(self.vertices[vi])
        glEnd()
        glDisable(GL_TEXTURE_2D)

    def draw(self):
        glPushMatrix()
        glTranslatef(-3, 0, 0)
        glRotatef(-45, 1, 1, 1)
        glScalef(0.5, 0.5, 0.5)
        self.draw_edges()
        glPopMatrix()
        glPushMatrix()
        glTranslatef(3, 0, 0)
        glRotatef(45, 1, 1, 1)
        glScalef(0.5, 0.5, 0.5)
        self.draw_polygons()
        glPopMatrix()
        self.draw_texture()


def main():
    pygame.init()
```

```
    size = (800, 600)
    screen = pygame.display.set_mode(size, OPENGL | HWSURFACE
| DOUBLEBUF)

    aspect = size[0] / size[1]
    gluPerspective(45, aspect, 0.1, 50)
    glTranslatef(0, 0, -10)

    glEnable(GL_LINE_SMOOTH)
    glEnable(GL_MULTISAMPLE)
    glEnable(GL_DEPTH_TEST)

    texture_id = load_texture('wall.jpg')

    cube = Cube(texture_id=texture_id)

    clock = pygame.time.Clock()
    is_rotate = True
    is_loop = True
    while is_loop:
        clock.tick(60)
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                is_loop = False
            if event.type == KEYUP and event.key == K_q:
                is_loop = False
            if event.type == KEYUP and event.key == K_SPACE:
                is_rotate = not is_rotate
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
        if is_rotate:
```

```
            glRotatef(1, 1, 3, 1)
        cube.draw()
        pygame.display.flip()

    pygame.quit()


main()
```



Пакет трехмерной графики Blender 3D

Введение

В настоящее время существует множество программ для работы с трехмерной графикой. Одной из наиболее популярных является Blender — свободно распространяемая платформа для разработки 3D-графики и анимации.

Широкий набор инструментов, а также возможность написания скриптов с помощью языка программирования Python делают Blender полноценным конкурентом для его коммерческих и индустриальных аналогов, и, кроме того, существование широкого комьюнити представляет собой серьёзное преимущество. Также он предоставляет возможность создания игр, работы с хромакеем и многое другое.

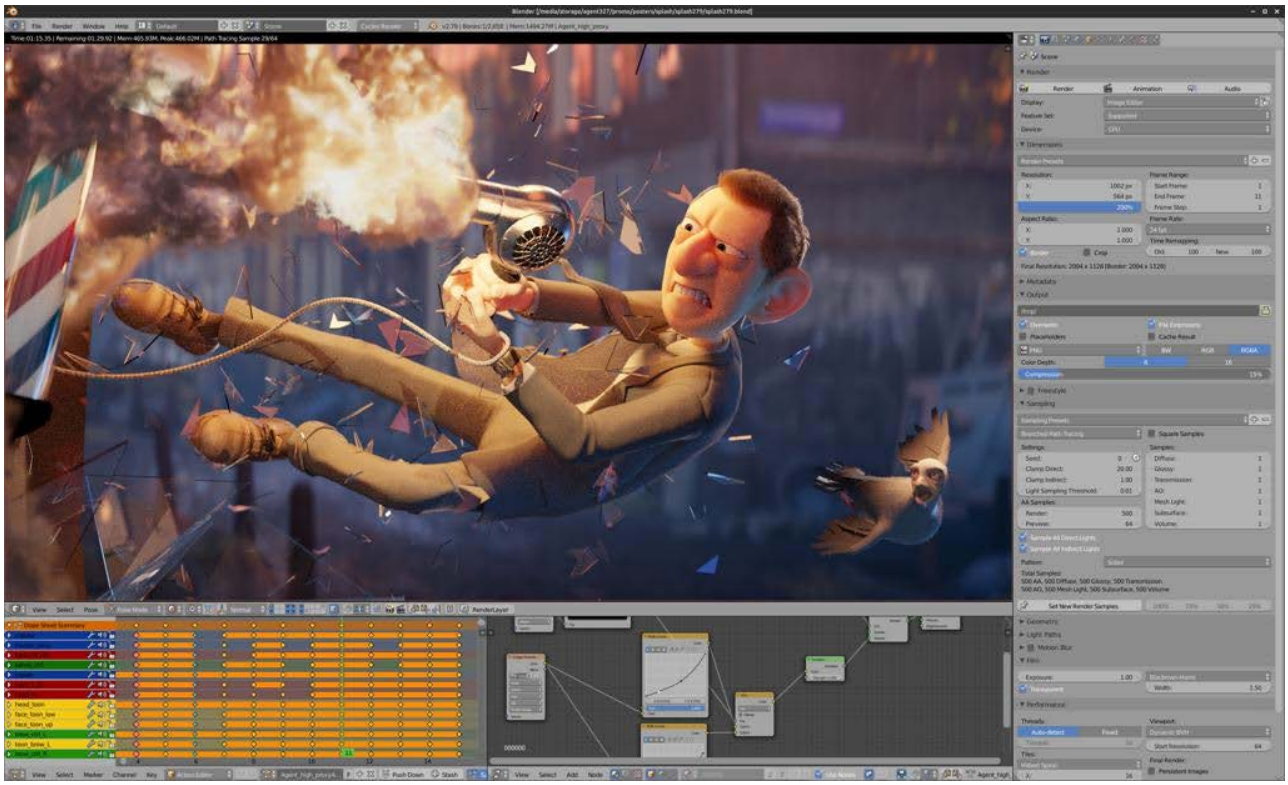


Рис. 1 Пример графики и интерфейса

Возможности

В целом пакет обладает следующим внушительным списком возможностей:

- Поддержка разнообразных геометрических примитивов, включая полигональные модели, систему быстрого моделирования в режиме subdivision surface (SubSurf), кривые Безье, поверхности NURBS, metaballs (метасферы), скульптурное моделирование и векторные шрифты.
- Универсальные встроенные механизмы рендеринга и интеграция с внешним рендерером YafRay, LuxRender и многими другими.
- Инструменты анимации, среди которых инверсная кинематика, скелетная анимация и сеточная деформация, анимация по ключевым кадрам, нелинейная анимация, редактирование весовых коэффициентов вершин, ограничители, динамика мягких тел (включая определение коллизий объектов при взаимодействии), динамика твёрдых тел на основе физического движка Bullet и система волос на основе частиц.
- Python используется как средство определения интерфейса, создания инструментов и прототипов, системы логики в играх, как средство импорта/экспорта файлов (например, COLLADA), автоматизации задач.
- Базовые функции нелинейного редактирования и комбинирования видео.

.blend

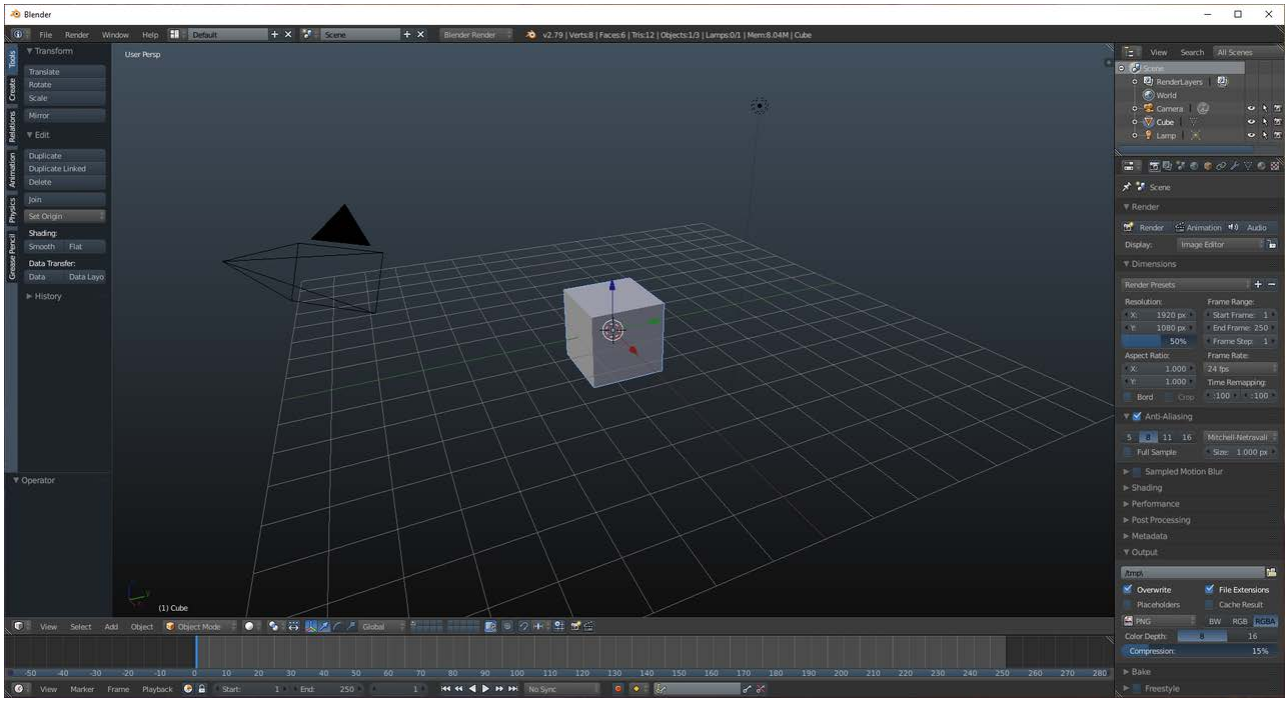
При разработке в Blender используется собственный формат файлов .blend, которые, кроме всего прочего, позволяют хранить несколько сцен в одном файле. Однако минусом является то, что эти файлы сложно конвертировать в другие форматы, по причине того, что он скорее представляет собой дамп области памяти программы, чем описание отношения объектов или информации о них.

Навигация

Ввиду сложности навигации и работы в 3D пространстве проектирования, навигация на первый взгляд может показаться нетривиальной, однако после некоторой практики становится вполне естественной. Существует несколько вариантов перемещения. Наиболее простым и интуитивным является использование сочетания Shift+F, это сочетание позволяет перемещаться в пространстве с помощью WASD и СКМ.

Если необходимо редактировать объекты, перемещение производится с помощью средней кнопки мыши, либо сочетания Alt+ЛКМ, которое можно

установить в настройках. Там же можно заменить способ выделения объектов с ПКМ на ЛКМ.



Комбинации клавиш

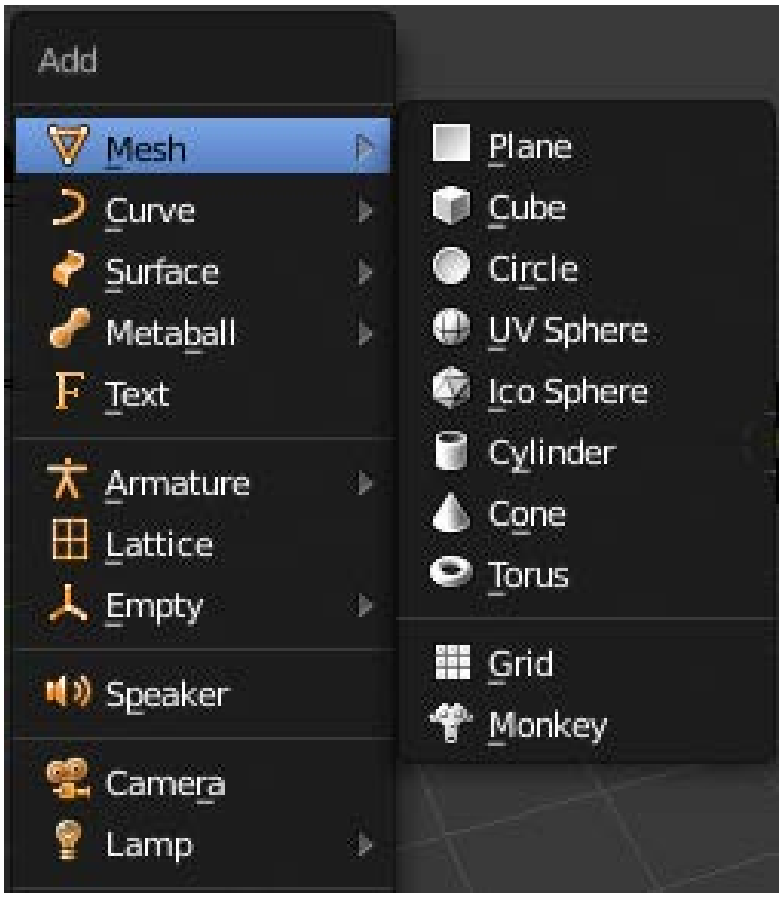
Вид	
Фронтальный	1 NumPad
Сверху	7 NumPad
Пользовательский Сверху	Sift+7_Num
Сбоку	3 NumPad
Перспектива (вкл/выкл)	5 NumPad
Камера (назначенная по-умочанию)	0 NumPad
Масштабировать отображение всех активных (выделеных) объектов сцены по размеру окна, и установить опорную точку поворота при изменении угла обзора сцены в центр выделения	.(Del) NumPad
Изменить угол обзора сцены	MMB+drag; 1-9

	NumPad
Установить в окне вид с четырех сторон (Quad View) одновременно	Ctrl+Alt+Q
Активный объект (камеру) неправить как текущий вектор вида на 3D сцену	Ctrl+Alt+NumPad_0
Объектный режим (манипуляции с объектами сцены)	
Добавить новый из библиотеки заготовок	Shift+A
Выбрать	RMB
Сдвинуть	G [x,y,z]
Повернуть	R [x,y,z]
Масштабировать	S [x,y,z]
Перенести в другой слой (вызов окна выбора слоя)	M
Выбрать всё	A
Удалить выбранное	X
Переключить в режим редактирования геометрии	Tab
Откат изменений	Ctrl+Z
Возврат после отката	Shift+Ctrl+Z
Дублировать (Duplicate) выделенные объекты	Shift+D
Объединить выделенные объекты в один	Ctrl+J
Сцена	
Вызов меню установки 3D-курсора	Shift+S
Разместить 3D-курсор в центре	Shift+C
Сохранить сцену в файл	Ctrl+S
Выбрать все объекты в активных слоях	A

Активировать слой (указать мышкой на нужную кнопку в панели слоев и щелкнуть)	LMB
Подключить указанный слой к активному	Shift+LMB
Указать слой для переноса активного объекта	M
Выделение прямоугольной областью	B
Выделение круглой областью	C
Изменение диаметра области выделения	Клесико
Выделить	LMB; drag
Снять выделение	MMB
Завершить выделение	Enter; RMB
Привязка (слежение) камеры к объекту (TrackTo Constraint). Для создания связи вначале правой кнопкой мыши выделяют камеру, потом (Shift+RMB) объект на который она будет направлена.	Ctrl+T
Рендер (Render) сцены	F12
Сохранить отрендеренную картинку в файл	F3
Окно	
Показать панель инструментов (Tool Shelf)	T
Показать панель параметров	N
Изменить режим отображения объектов	Z
Установить в окне вид с четырех сторон (Quad View) одновременно	Ctrl+Alt+Q
Манипуляции с арматурой (скелетом)	
Выровнять оси вращения соединенных звеньев	Ctrl+N
Зеркальное отражение выделенных звеньев (костей)	Ctrl+M [x,y,z]

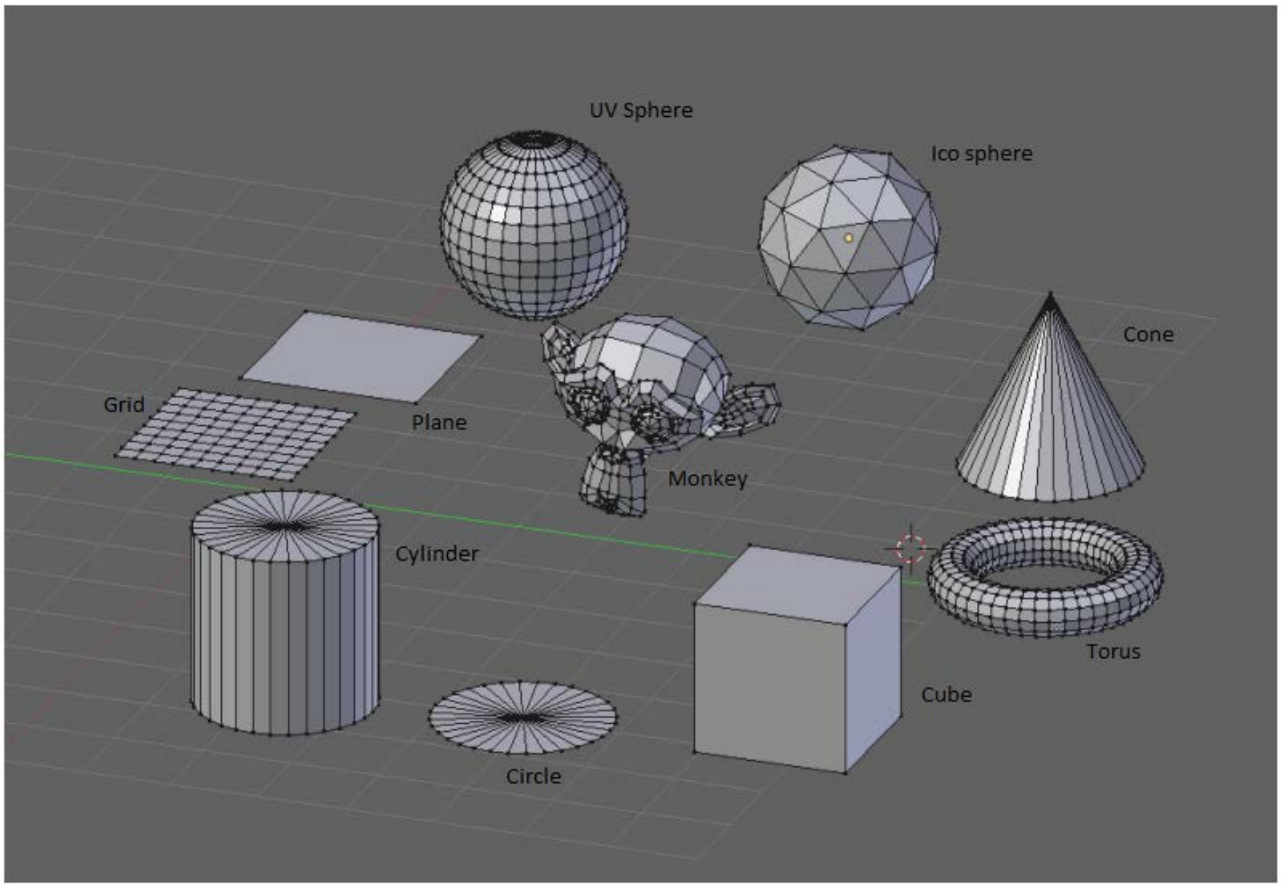
Назначить арматуру "родителем" меша (вначале выделяется меш, затем с Shift арматура)	Ctrl+P
Переключиться в режим позы	Ctrl+Tab
Вернуть арматуру в исходную позицию (очистить перемещение)	Alt+G
Очистить вращение	Alt+R
Меню Сохранение позы в кадре	I
Скопировать позу (положение) арматуры из текущего кадра анимации в буфер обмена	Ctrl+C
Вставить отраженную позу (X-Flipped Pose) из буфера обмена в кадр	Ctrl+Shift+V
Вставить в кадр позу из буфера обмена	Ctrl+V
Просмотреть анимацию	Alt+A
Создать новое звено до позиции курсора	Ctrl+LMB
Подменю настройки звеньев	W
Выделить верхнее/нижнее от текущего звено арматуры	] / [
Создать звено от выделенной вешины до 3D курсора	F
Отделить выделенные звенья в отдельный скелет	Ctrl+Alt+P
Объединить два скелета в один	Ctrl+J
Выделить все присоединенные звенья одной линии	L
Включить связь между звеньями	Ctrl+P
Установить текущее расположение звеньев скелета как исходное (as Rest Pose)	Ctrl+A
Выровнять направление выделенных звеньев	Ctrl+Alt+A

Примитивы



Выбор примитивов в меню создания Shift+A

Выбор примитивов в боковом меню

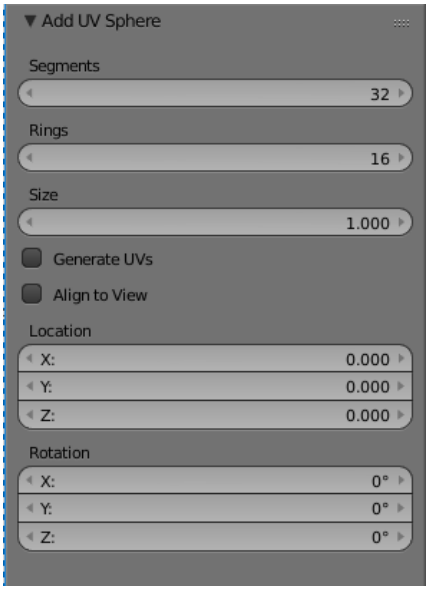


Mesh — полисетка. Структура полисетки состоит из следующих элементов:

- вершины (vertices),
- ребра (edges),
- грани (faces).

Среди примитивов есть двумерные объекты: плоскость (plane), сетка (grid) и окружность (circle). Остальные фигуры трехмерные.

При создании объекта в левом меню появляется раздел Add...
В нем можно задавать базовые характеристики объектов: радиус, начальные координаты, разрешение сетки и т.д.
Важно помнить, что как только над объектом будет произведено какое-либо действие (в том числе перемещение) это меню исчезнет.
Также не обязательно задавать высокое разрешение для объектов, так как это может привести к сложности обработки. Сглаживание проще произвести с помощью модификатора Subdivision Surface для стадии рендеринга.



Опции создания сферы

Generate UV требуется для наложения текстур. Align to View разворачивает объект в соответствии с точкой зрения пользователя.

Булевы операции

Меню модификатора Boolean

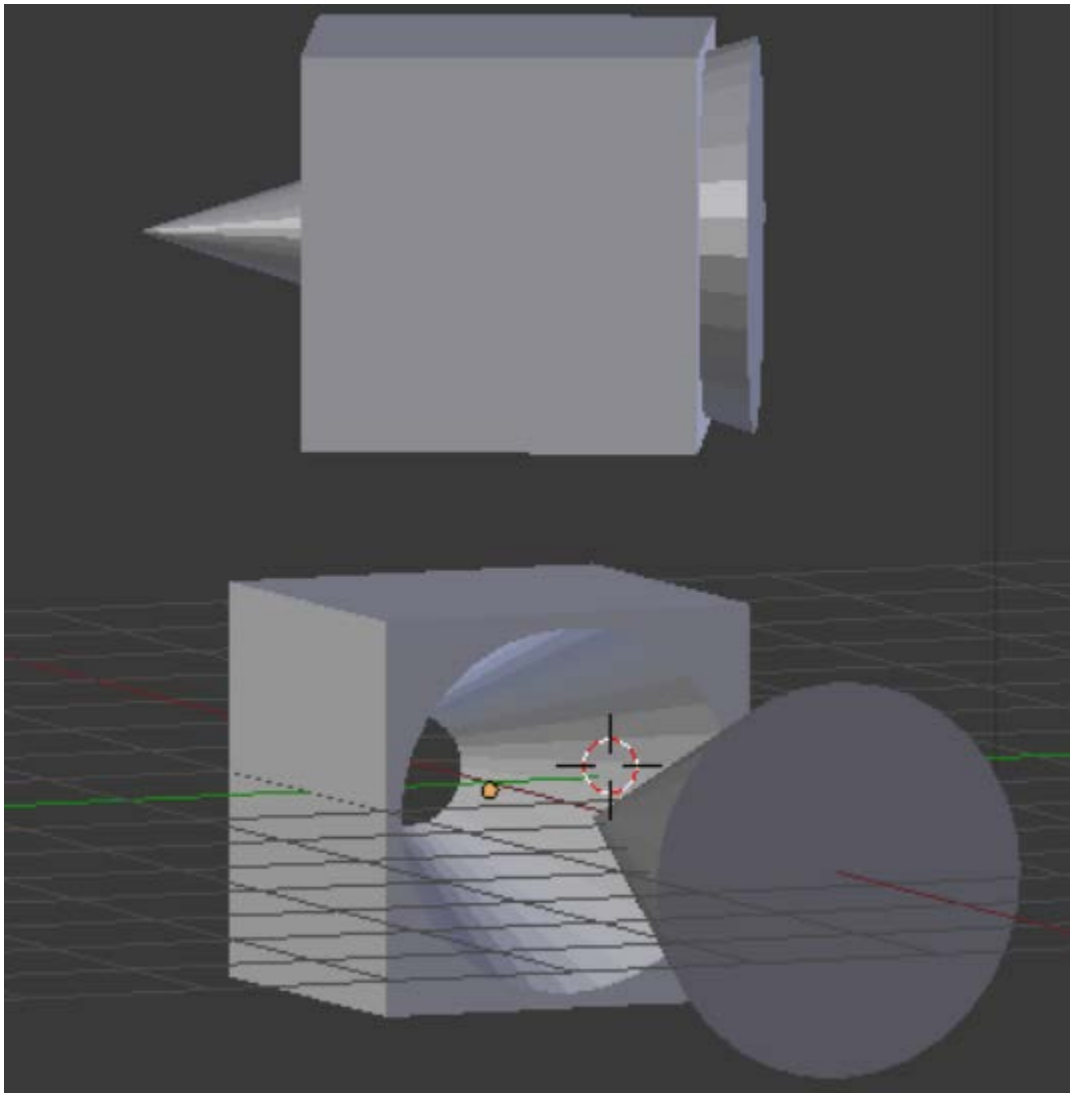
Для объектов вида Mesh доступен такой инструмент как модификатор Boolean. Он описывает взаимодействие двух объектов и создания на их основе смешанного результата, что может серьезно упростить некоторые задачи моделирования. Для использования Boolean нужно лишь присоединить необходимый спецификатор к одному из объектов.

Настройки модификатора Boolean:

- меню Operation (Операция) позволяет выбрать способ взаимодействия двух примитивов:
  - Intersect (Пересечение);
  - Union (Объединение);

    - Difference (Различие);
- поле Object (Объект) — название второго примитива для действия. При щелчке появляется меню, где можно выбрать объект из списка.

К примеру, операция Difference может позволить сделать в кубе отверстие в форме конуса:



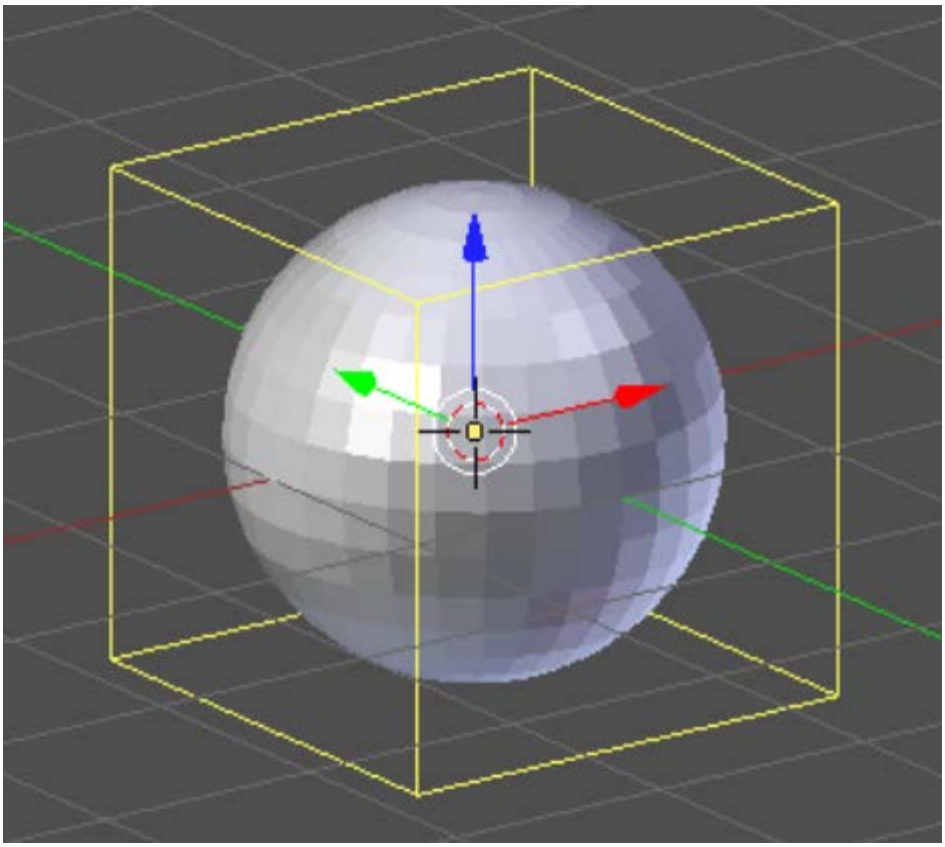
Вспомогательная решетка

Lattice (Решетка) — это вспомогательный объект, который не отображается при обработке сцены, но позволяет деформировать модель. Для использования Lattice нужно выполнить следующее:

1. Добавить в сцену сам объект из меню Add | Lattice.
2. Добавить модификатор Lattice к модели, которую нужно деформировать.

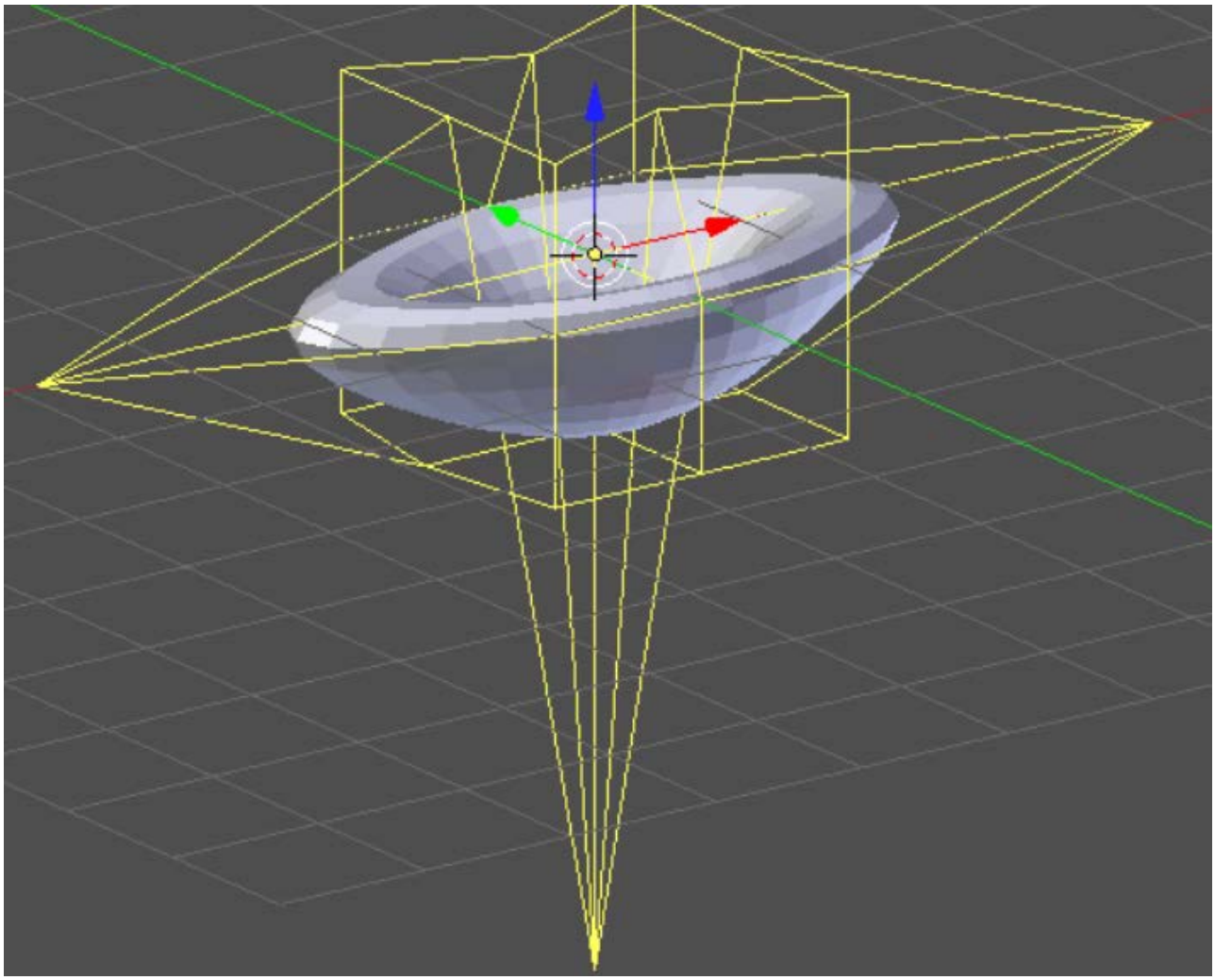
3. Выбрать в модификаторе имеющийся в сцене объект Lattice.

Главным условием работы решетки Lattice является местонахождение модели внутри нее. По умолчанию Lattice выглядит, как обычный куб с 12 ребрами

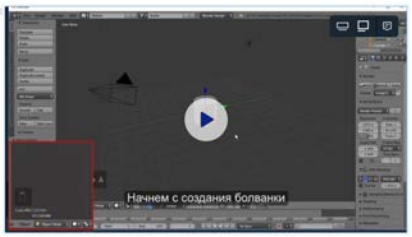


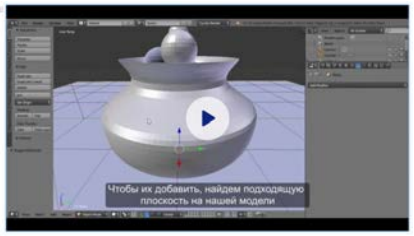
Решетка подчиняется тем же правилам манипуляций, что и другие объекты. Рассмотрим панель модификатора Lattice. Всего там находится две опции:
Object (Объект) — при щелчке по полю откроется меню, где можно выбрать из списка объект Lattice;
Vertex Group (Группа вершин) — если модель имеет созданные группы вершин, то данное поле позволяет выбрать из них. В этом случае Lattice будет работать только с группой.
В отличие от обычных примитивов доступно перемещение, масштабирование и ротация только вершин. Собственно, манипуляции с вершинами Lattice влияют на деформацию модели. Простое моделирование с Mesh. Объект Lattice имеет свои собственные настройки, которые доступны в окне Properties.

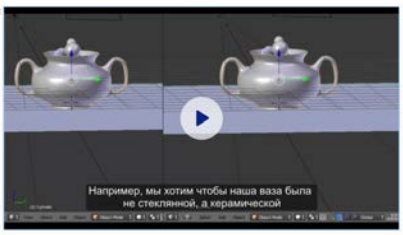
В некоторых случаях восьми точек Lattice бывает недостаточно для изменения модели. В настройках объекта можно изменить разбивку структуры с помощью параметров U, V, W. Рядом с этими полями находятся меню, где можно выбрать алгоритм, по которому Lattice будет деформировать модель. Вот, например, преобразование сферы с помощью параметра BSpline:



Создание сцены и рендеринг (видеоуроки)





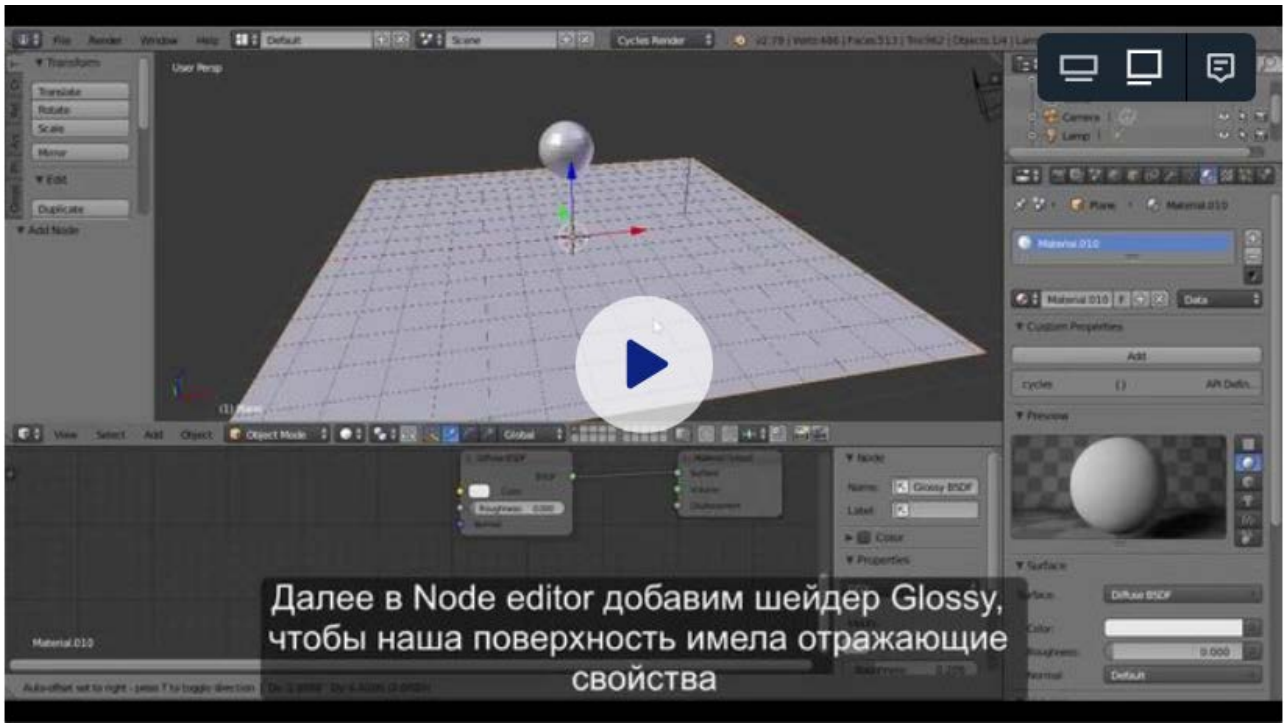


https://www.dropbox.com/s/95on7h990k5j9vb/BLENDER1.mp4?dl=0

https://www.dropbox.com/s/olnhcdu3ysyps41/BLENDER2.mp4?dl=0

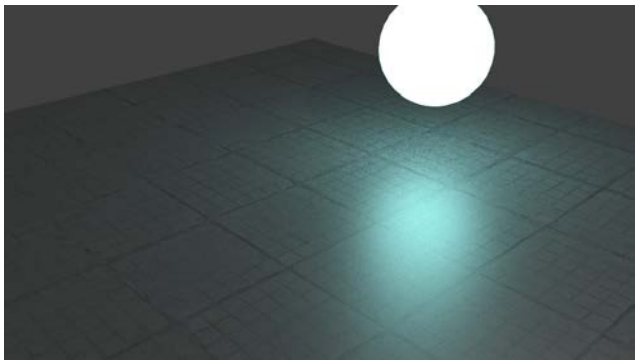
https://www.dropbox.com/s/b5uhjj097irnytc/Blender3.mp4?dl=0

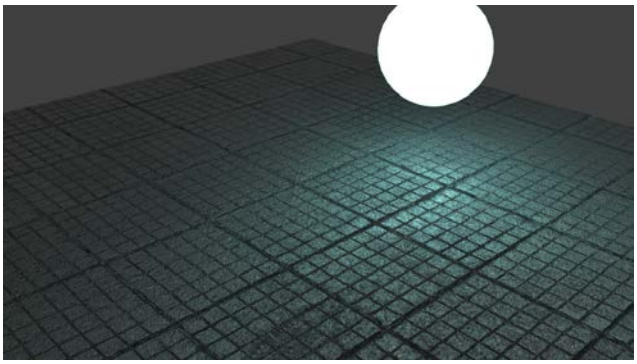
Текстуры (видеоурок)



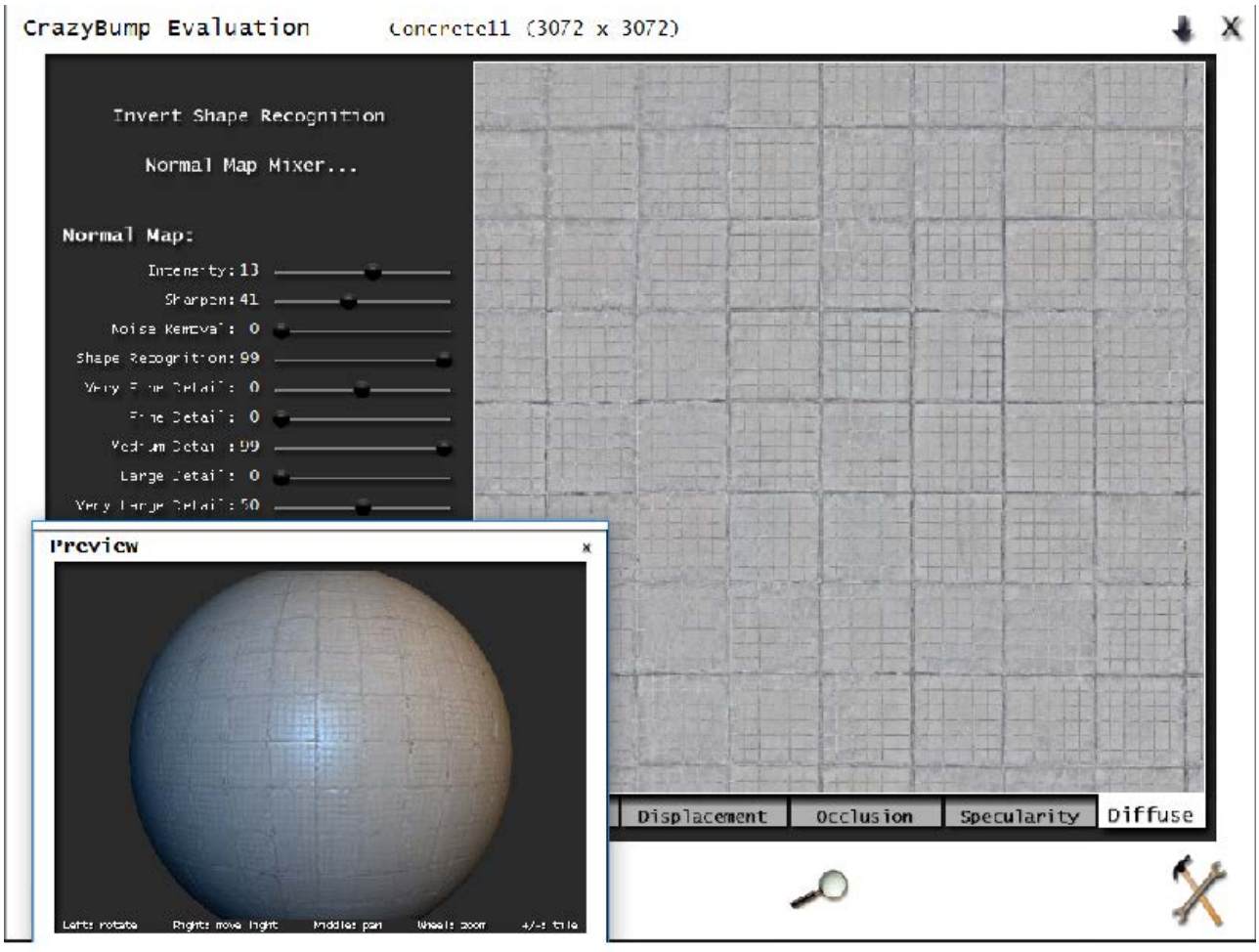
https://www.dropbox.com/s/fox54pc1o3iadqb/BLENDER4.1.mp4?dl=0

Разница между текстурой без использования всомогательных карт и с ними:





Для создания карт существует платная программа CrazyBump, позволяющая создавать и редактировать карты текстуры.

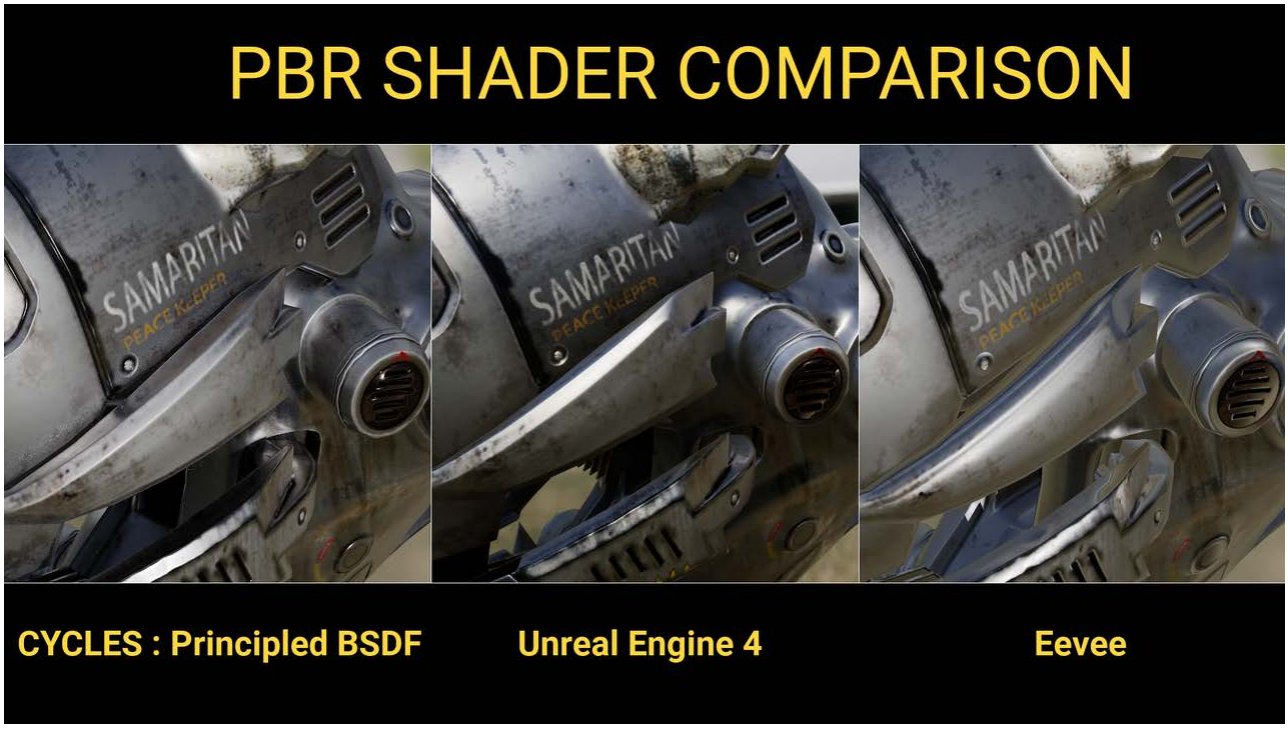


Рендер в реальном времени

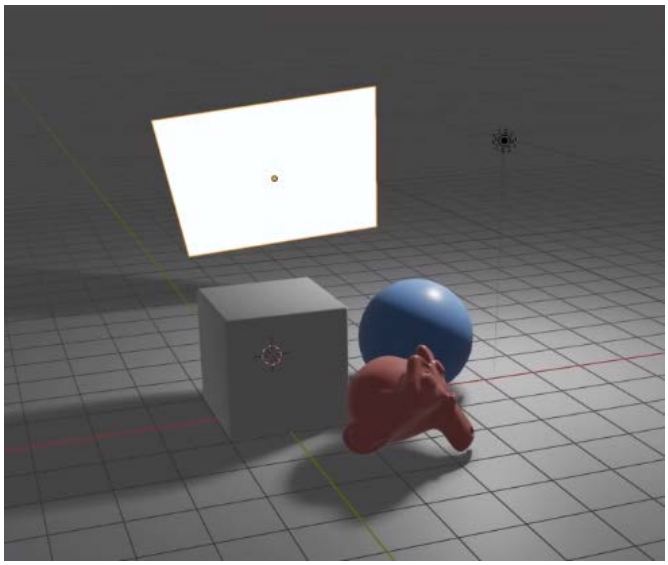
Новая версия Blender 2.8 среди прочих нововведений получил новый движок для рендера — Eevee. Его главной особенностью является возможность рендера

в реальном времени. Столь высокая скорость обработки достигается за счет упрощений в расчетах конечного изображения, которые также используются в игровых движках.

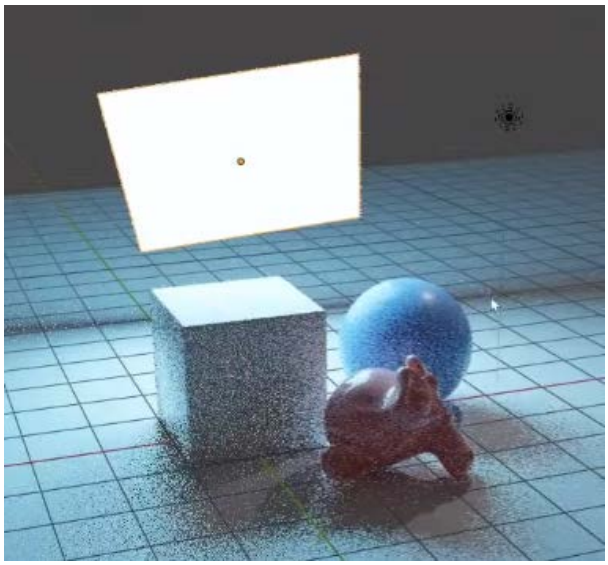
Однако из-за этих упрощений, которых нет в Cycles, добиться того же эффекта фотореализма на данный момент нереально.



Одним из применяемых упрощений является замена расчета мягкой тени на “размытую” тень, так как расчет мягкой тени является очень сложной и затратной задачей.
Также некоторые эффекты, например Emission, не работают в этом режиме, так как необходимые упрощения не дают их рассчитать.



Рендер в реальном времени с помощью Eevee



Рендер в реальном времени с помощью Cycles

Такой рендер без “шума” в реальном времени позволяет ускорить работу с моделями, упрощая просмотр текстур и прочего, а также может значительно уменьшить временные затраты на рендер в проектах, где не требуется фотореализм.

В настоящий момент этот движок находится в активной разработке и, возможно, в будущем его возможности значительно расширятся.

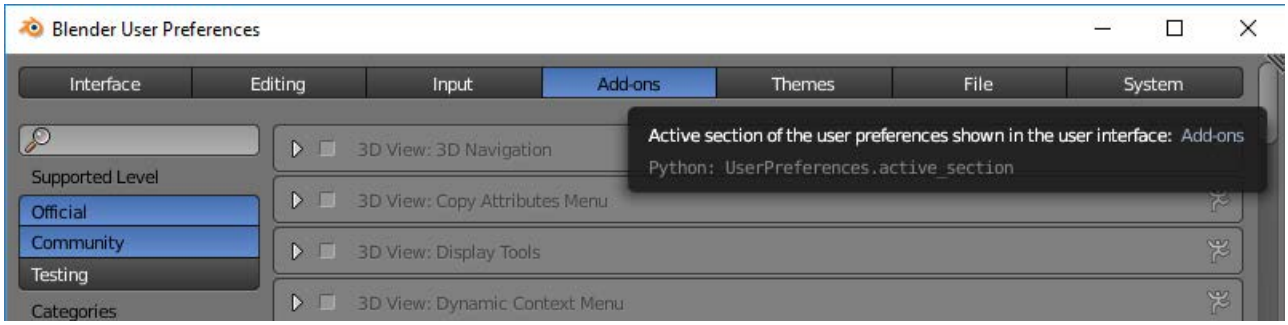
https://code.blender.org/2018/03/eevee-f-a-q/

EEVEE F.A.Q. — Blender Developers Blog • code.blender.org

Python API

Одной из мощных возможностей Blender'a является взаимодействие с интерфесом с помощью языка программирования Python, что позволяет контролировать почти все аспекты Blender'a. Например, вы можете написать скрипт, которыый позволит создавать прецедурально сгенерированные текстуры или реализовать экспорт/импорт мешей. Также возможно процедуральное создание анимаций или модификация уже существующих сцен. Кроме этого вы можете создать для своей надстройки удобный для вас интерфейс и поделиться вашим инструментом со всем коммьюнити.

В целом, скрипт, который расширяет возможности Blender'a называется аддоном. Включать/выключать существующие аддоны можно в пользовательских настройках(User Preferences).



Модуль BPY

BPY связывает Python и Blender. В этом модуле содержаться несколько подмодулей. Вот несколько часто используемых:

- `bpy.data` — связан с содержанием активного документа.
- `bpy.types` —содержит информацию о типах объектов в `bpy.data`.
- `bpy.ops` — содержит операторы, которые связаны с функциями самого Blender'a. Их можно связать с хоткеями, элементами меню или кнопками. В этом подмодуле в большинстве случаев определяются новые операторы. каждый оператор должен обладать уникальным именем.
- `bpy.context` — содержит настройки 3D режима, выбранных объектов и так далее. Также этот подмодуль доступен в консоли через глобальную переменную `C`.
- `bpy.props` — содержит функции, через которые можно задать свойства. Позволяет скрипту задавать информацию сцене, которые, например, могут изменяться пользователем с помощью элементов интерфейса, для контроля поведения скрипта.

Модуль mathutils

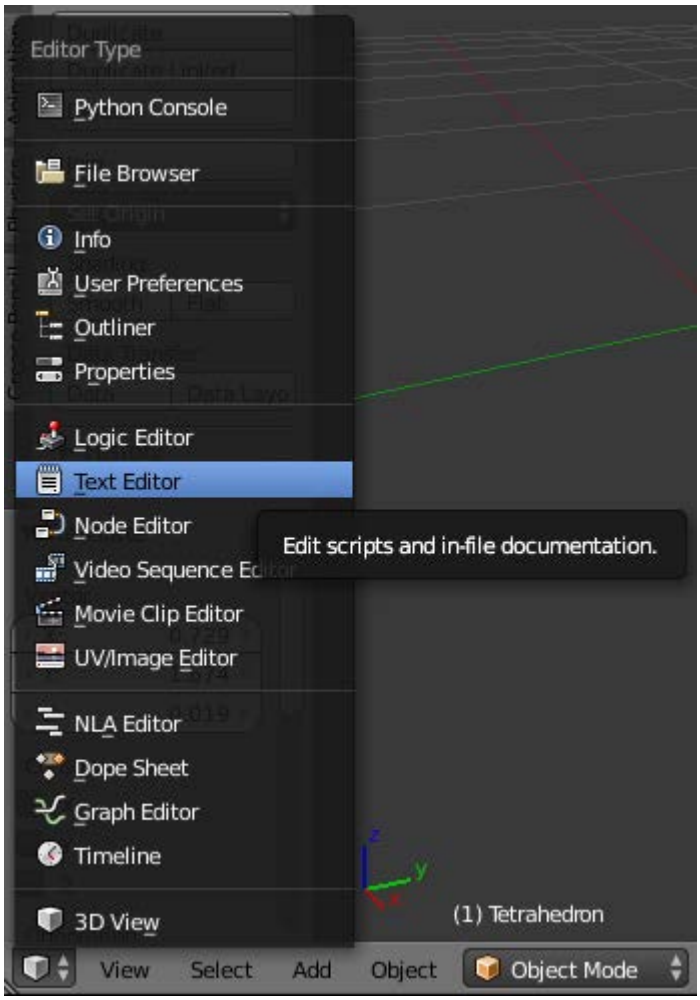
Этот модуль определяет множество важных классов, которые активно используются всем остальным Blender API.

- `Vector`: представление двухмерных или трехмерных координат.
- `Matrix`: самый общий способ представления любого вида лтинейной трансформации.
- `Euler`: дает способ представления поворота объекта как набора углов Эйлера, которые представляют собой поворот на некоторый угол

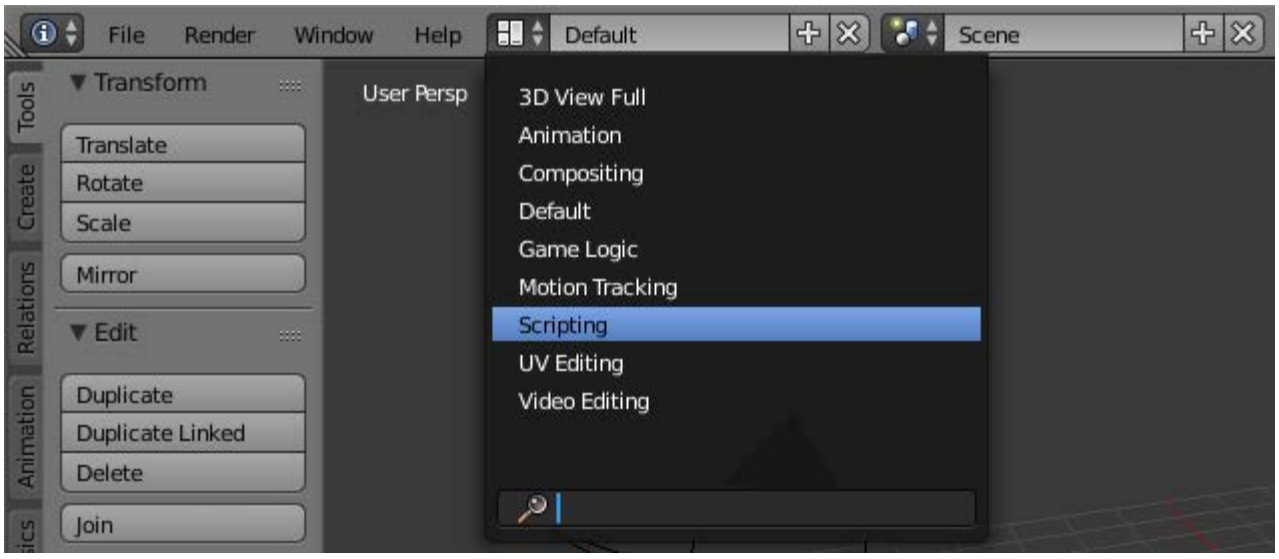
относительно осей X, Y и Z. В нем содержится широко известная ловушка из гироскопии — **Складывание рамок**, или **блокировка карданного подвеса**
- `Quaternion`: более сложный математически способ определения поворотов. Обладает своими преимуществами, например, отсутствием складывания рамок или полее гладуой интерполяцией двух произвольных поворотов(что важно для анимации персонажа).
- `Color`: представление цветов RGB и преобразования к/из HSV пространству (без альфа канала).

Первые шаги

Попробуем сами создать скрипт.
Откройте новый документ Blender'a. В удобном вам окне откройте вкладку Text Editor.



Или в выпадающем меню раскладки выберете вариант Scripting.



Прежде чем начать редактирование текста, нужно создать новый текстовый блок. Для этого нажмите на кнопку New.

В отличие от консолького окна, здесь пакеты не встроены явно. Так что, как и в любом скрипте на Питоне, вам необходимо явно обращаться к каждому модулю. Ваш оператор будет определен как подкласс подкласса bpy.types.Operator. Он должен иметь атрибут класса bl_idname, который задаёт имя оператора и bl_label задает имя видимое пользователю в среде Blender'a.
Давайте создадим оператор, который будет создавать тетраэдр. Класс оператора примет следующий вид:

```
class MakeTetrahedron(bpy.types.Operator) :
  bl_idname = "mesh.make_tetrahedron"
  bl_label = "Add Tetrahedron"
```

Также должен быть определен метод вызова.

```
def invoke(self, context, event) :
```

Метод invoke как раз и отвечает за выполнение самой функции оператора. После завершения работы, он возвращает значение, которое состоит из набора строк, которые указывают Blender'y на состояние оператора. Чтобы обозначить успешное выполнение метода окончим invoke строкой

```
return {"FINISHED"}
```

А теперь самое сложное: обозначение тетраэдра. Нужно передать координаты вершин. Со сторонами, соответствующими по длине 1 мере Blender'а, эти значения запишется следующим образом:

```
Vertices = \
[
  mathutils.Vector((0, -1 / math.sqrt(3),0)),
  mathutils.Vector((0.5, 1 / (2 * math.sqrt(3)), 0)),
  mathutils.Vector((-0.5, 1 / (2 * math.sqrt(3)), 0)),
  mathutils.Vector((0, 0, math.sqrt(2 / 3))),
]
```

Теперь зададим блок меша с именем “Tetrahedron”

```
NewMesh = bpy.data.meshes.new("Tetrahedron")
```

Заполним его уже заданными нами вершинами и соответствующими граням плоскостями:

```
NewMesh.from_pydata \
  (
    Vertices,
    [],
    [[0, 1, 2], [0, 1, 3], [1, 2, 3], [2, 0, 3]]
  )
```

Метод NewMesh.from_pydata получает первым аргументом массив векторов определяющих вершины, второй аргумент-массив опередений ребер и третий аргумент-массив определений граней. Заметьте, что передавать нужно или грани или ребра, НО не все сразу, что-то одно нужно передать пустым списком.

Кроме всго прочего, необходимо сообщить Blender'у об обновлении меша:

```
NewMesh.update()
```

Теперь зададим блок объекта и укажем, что он относится к мешу, который мы создаем NewMesh:

```
NewObj = bpy.data.objects.new("Tetrahedron", NewMesh)
```

Объект не появится для пользователя, пока не будет связан с активной сценой.

```
context.scene.objects.link(NewObj)
```

Вот и все, что необходимо для создания скрипта, который будет создавать новый тетраэдр.
Полный код скрипта:

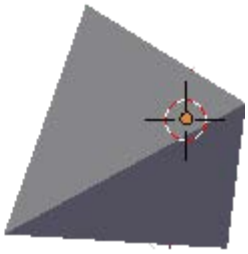
```
import math
import bpy
import mathutils

class MakeTetrahedron(bpy.types.Operator) :
  bl_idname = "mesh.make_tetrahedron"
  bl_label = "Add Tetrahedron"

  def invoke(self, context, event) :
    Vertices = \
      [
        mathutils.Vector((0, -1 / math.sqrt(3),0)),
        mathutils.Vector((0.5, 1 / (2 * math.sqrt(3)), 0)),
        mathutils.Vector((-0.5, 1 / (2 * math.sqrt(3)), 0)),
        mathutils.Vector((0, 0, math.sqrt(2 / 3))),
      ]
    NewMesh = bpy.data.meshes.new("Tetrahedron")
    NewMesh.from_pydata \
      (
        Vertices,
        [],
        [[0, 1, 2], [0, 1, 3], [1, 2, 3], [2, 0, 3]]
      )
    NewMesh.update()
    NewObj = bpy.data.objects.new("Tetrahedron", NewMesh)
    context.scene.objects.link(NewObj)
    return {"FINISHED"}
  #end invoke
```

```
#end MakeTetrahedron


bpy.utils.register_class(MakeTetrahedron)
```

Следующая строчка в конце нужна для того, чтобы наш оператор создания тетраэдра появился среди встроенной коллексции Blender'a.

```
bpy.utils.register_class(MakeTetrahedron)
```

Для запуска скрипта нажмите Alt+P. После запуска наш оператор зарегистрируется среди остальных и будет готов к использованию. Если вы совершили какие-либо синтаксические ошибки Blender сообщит вам об этом в новом окошке. исправите их и снова попробуйте запустить скрипт.
Для создания нашего объекта теперь нужно лишь в области сцены нажать на пробел и в появившемся окне начать набирать имя оператора, В нашем случае - “Add tetrahedron”.



Создание панели

Чтобы полноценно использовать наш скрипт, следует создать для него интерфейс, который позволит не искать его через пробел.
Создадим наш интерфейс в Панели инструментов(Tool Shelf), которую можно вызвать нажав в режиме 3D просмотра 'T'.
Панели определяются подклассом bpy.types.Panel. Нужно определить атрибуты bl_space_type, bl_region_type, bl_categoryand bl_context, чтобы задатьь контекст появления элемента интерфейса.

```
class TetrahedronMakerPanel(bpy.types.Panel):
    bl_space_type = "VIEW_3D"
```

```
    bl_region_type = "TOOLS"
    bl_context = "objectmode"
    bl_category = "Create"
    bl_label = "Add Tetrahedron"
```

Также нужно задать метод draw, который определяет что будет отображаться в панели.

```
def draw(self, context):
  TheCol = self.layout.column(align=True)
  TheCol.operator("mesh.make_tetrahedron", text="Add Tetrahedr
on")
#end draw
```

Конечно же необходимо, чтобы действия,, совершаемые по клику на элементы этой панели, можно было отменить. Для этого нужно подписать:

```
bl_options = {"UNDO"}
```

Это позволить отменять действия с помощью ctrl+Z.

Таким образом весь наш скрипт примет вид:

```
import math
import bpy
import mathutils

class TetrahedronMakerPanel(bpy.types.Panel):
  bl_space_type = "VIEW_3D"
  bl_region_type = "TOOLS"
  bl_context = "objectmode"
  bl_category = "Create"
  bl_label = "Add Tetrahedron"

  def draw(self, context):
    TheCol = self.layout.column(align=True)
    TheCol.operator("mesh.make_tetrahedron", text="Add Tetrahe
dron")
```

```
  #end draw

#end TetrahedronMakerPanel

class MakeTetrahedron(bpy.types.Operator):
  bl_idname = "mesh.make_tetrahedron"
  bl_label = "Add Tetrahedron"
  bl_options = {"UNDO"}

  def invoke(self, context, event):
    Vertices = \
    [
      mathutils.Vector((0, -1 / math.sqrt(3),0)),
      mathutils.Vector((0.5, 1 / (2 * math.sqrt(3)), 0)),
      mathutils.Vector((-0.5, 1 / (2 * math.sqrt(3)), 0)),
      mathutils.Vector((0, 0, math.sqrt(2 / 3))),
    ]
    NewMesh = bpy.data.meshes.new("Tetrahedron")
    NewMesh.from_pydata \
    (
      Vertices,
      [],
      [[0, 1, 2], [0, 1, 3], [1, 2, 3], [2, 0, 3]]
     )
     NewMesh.update()
     NewObj = bpy.data.objects.new("Tetrahedron", NewMesh)
     context.scene.objects.link(NewObj)
     return {"FINISHED"}
  #end invoke
```

```
#end MakeTetrahedron


bpy.utils.register_class(MakeTetrahedron)
bpy.utils.register_class(TetrahedronMakerPanel)
```

Заметьте, что мы дополнительно вызываем register_class и для панели.
Как и в прошлый раз, после нажатия Alt+P, ничего не произойдет. Blender обработает определения подклассов и зарегистрирует их в соответствующих местах.
Но, если теперь перейти к нашей сцене, Object Mode, нажмите T, чтобы вызвать панель инструментов, вы увидите новую панель

Задание свойства

Свойства необходимо задавать на той же стадии, на которой происходит регистрация наших классов. Удобнее будет собрать всё это вместе в одном методе. Назовем его register.

```
def register() :
  bpy.utils.register_class(MakeTetrahedron)
  bpy.utils.register_class(TetrahedronMakerPanel)
  bpy.types.Scene.make_tetrahedron_inverted = bpy.props.BoolPr
operty \
    (
      name = "Upside Down",
      description = "Generate the tetrahedron upside down",
      default = False
    )
#end register
```

Мы добавляем свойство как новый атрибут сцены. Используем bpy.props.BoolProperty для создания поля флажка.
Чтобы наш флажок появился необходимо добавиить следующую сторку к draw методу:

```
TheCol.prop(context.scene, "make_tetrahedron_inverted")
```

Для корректного функционирования аддона при публикации необходимо создать метод, который будет отменять изменения вносимые скриптом.

```
def unregister() :
    bpy.utils.unregister_class(MakeTetrahedron)
    bpy.utils.unregister_class(TetrahedronMakerPanel)
    del bpy.types.Scene.make_tetrahedron_inverted
#end unregister
```

В конце нашего скрипта нужно добавить

```
if __name__ == "__main__" :
    register()
#end if
```

чтобы вызвать регистрацию в случае, если Blender не сделает этого сам.
Для задания самого свойства нужно использовать его в методе execute.

```
Scale = -1 if context.scene.make_tetrahedron_inverted else 1
```

Этим мы инвертируем положение по Z координате.
Результат:

```
import math
import bpy
import mathutils

class TetrahedronMakerPanel(bpy.types.Panel) :
  bl_space_type = "VIEW_3D"
  bl_region_type = "TOOLS"
  bl_context = "objectmode"
  bl_category = "Create"
  bl_label = "Add Tetrahedron"

  def draw(self, context) :
    TheCol = self.layout.column(align = True)
    TheCol.prop(context.scene, "make_tetrahedron_inverted")
```

```
    TheCol.operator("mesh.make_tetrahedron", text = "Add Tetra
hedron")
  #end draw

#end TetrahedronMakerPanel

class MakeTetrahedron(bpy.types.Operator) :
  bl_idname = "mesh.make_tetrahedron"
  bl_label = "Add Tetrahedron"
  bl_options = {"UNDO"}

  def invoke(self, context, event) :
    Scale = -1 if context.scene.make_tetrahedron_inverted else
1
    Vertices = \
      [
        mathutils.Vector((0, -1 / math.sqrt(3),0)),
        mathutils.Vector((0.5, 1 / (2 * math.sqrt(3)), 0)),
        mathutils.Vector((-0.5, 1 / (2 * math.sqrt(3)), 0)),
        mathutils.Vector((0, 0, Scale * math.sqrt(2 / 3))),
      ]
    NewMesh = bpy.data.meshes.new("Tetrahedron")
    NewMesh.from_pydata \
    (
      Vertices,
      [],
      [[0, 1, 2], [0, 1, 3], [1, 2, 3], [2, 0, 3]]
    )
    NewMesh.update()
    NewObj = bpy.data.objects.new("Tetrahedron", NewMesh)
```

```
    context.scene.objects.link(NewObj)
    return {"FINISHED"}
  #end invoke

#end MakeTetrahedron

def register() :
  bpy.utils.register_class(MakeTetrahedron)
  bpy.utils.register_class(TetrahedronMakerPanel)
  bpy.types.Scene.make_tetrahedron_inverted = bpy.props.BoolPr
operty \
    (
      name = "Upside Down",
      description = "Generate the tetrahedron upside down",
      default = False
    )
#end register

def unregister() :
  bpy.utils.unregister_class(MakeTetrahedron)
  bpy.utils.unregister_class(TetrahedronMakerPanel)
  del bpy.types.Scene.make_tetrahedron_inverted
#end unregister

if __name__ == "__main__" :
  register()
#end if
```

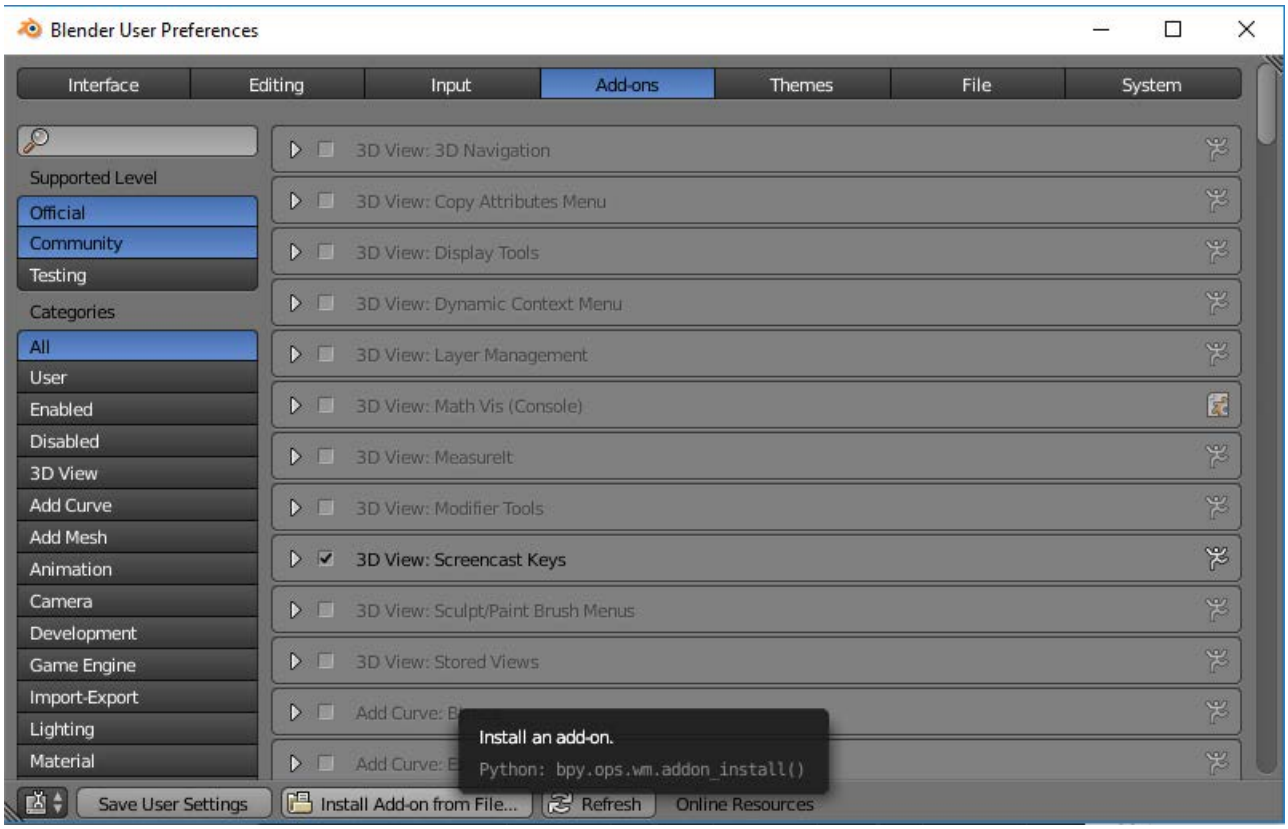
Самостоятельный аддон

Для того, чтобы сделать скрипт устанавливаемым и доступным для распространения аддоном нужно задать для него глобальный параметр bl_info.

```
bl_info = \
    {
        "name" : "Tetrahedron Maker",
        "author" : "YourName <yourmail@example.com>",
        "version" : (1, 0, 0),
        "blender" : (2, 5, 7),
        "location" : "View 3D > Edit Mode > Tool Shelf",
        "description" :
            "Generate a tetrahedron mesh",
        "warning" : "",
        "wiki_url" : "",
        "tracker_url" : "",
        "category" : "Add Mesh",
    }
```

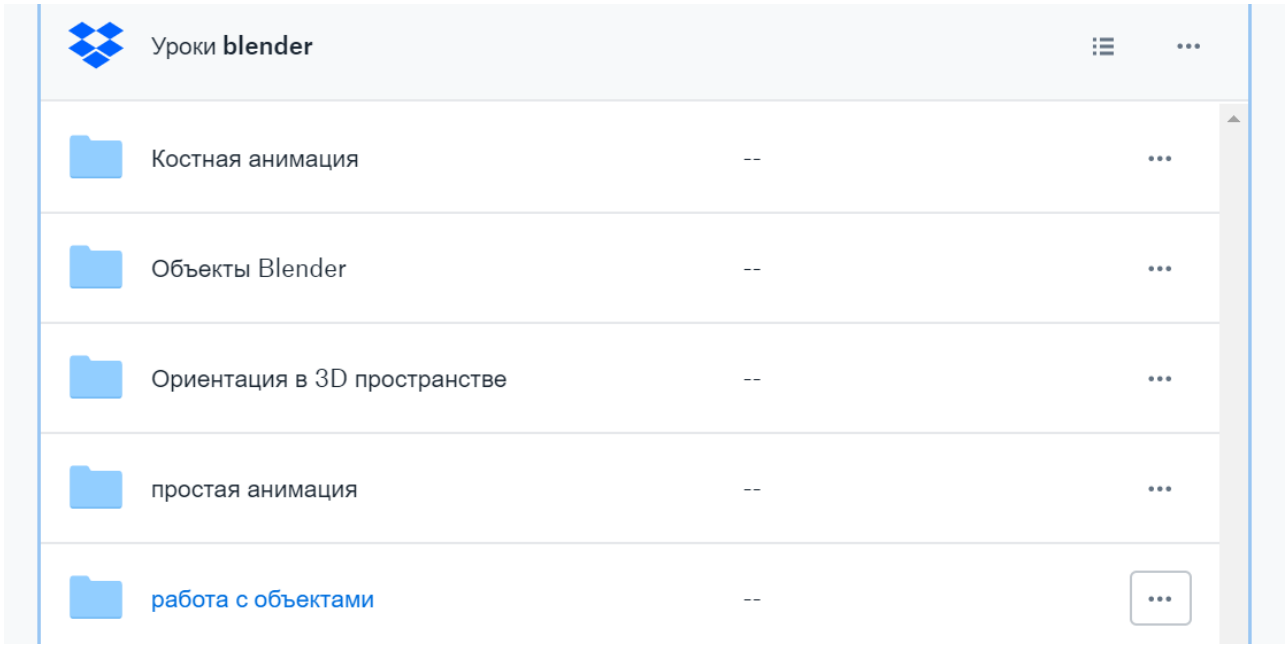
После этого скрипт готов стать полноценным аддоном. Сохраним скрипт нажав Save As и задав ему имя. Например, TetrahedronMaker.py.

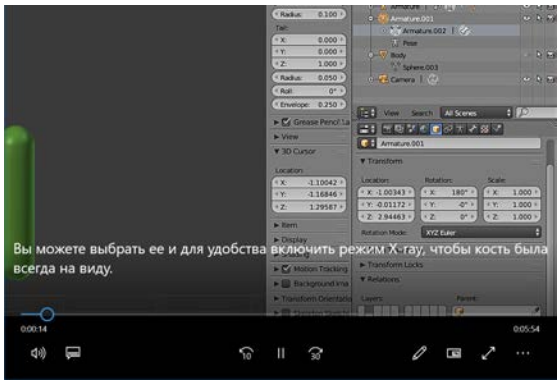
Теперь он будет доступен для установки с помощью: User Preferences→Add-ons→Install Add-on from File...

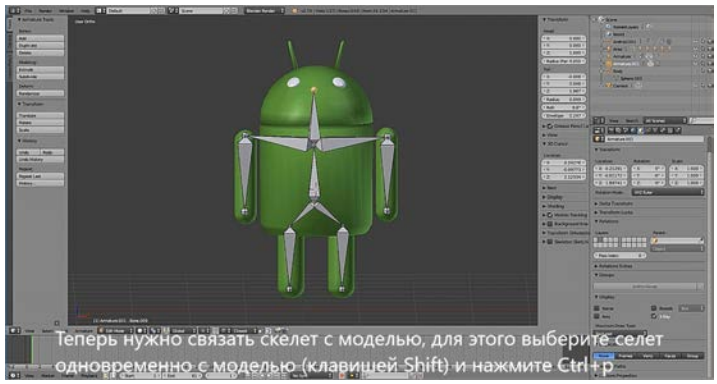


Разные видеоуроки

Основной функционал

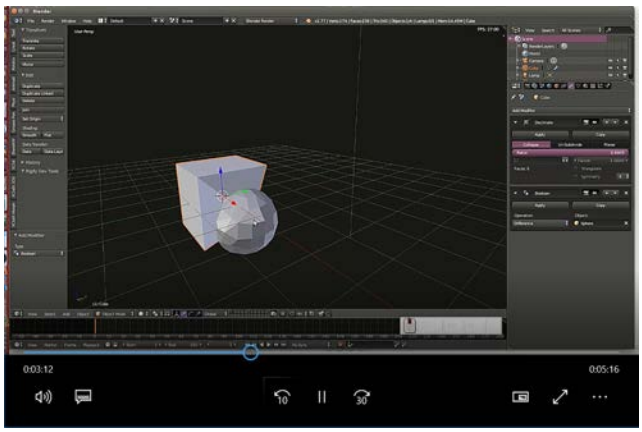


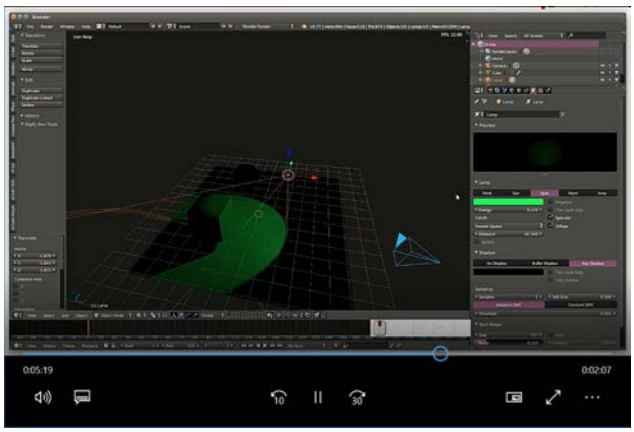




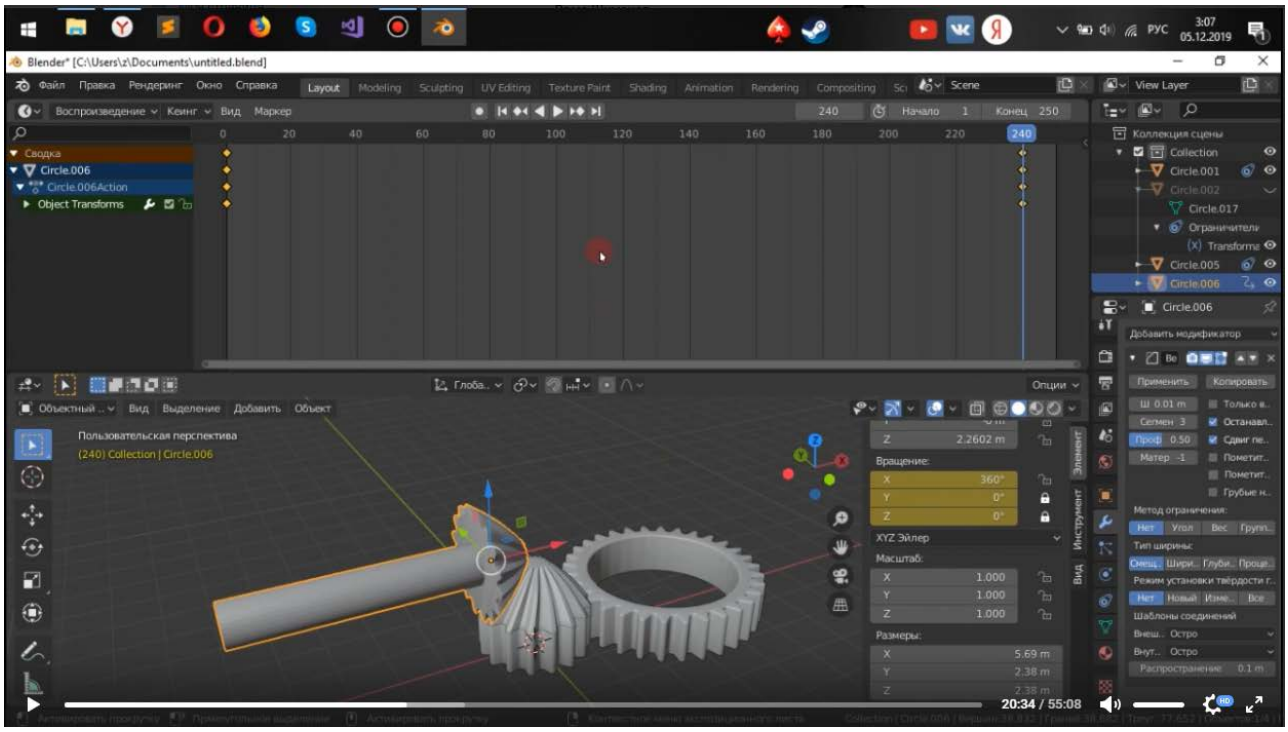
Озвученные уроки Blender

Моделирование	Скрипты (BPY)
• blender-1-interface.mp4 • blender-2-objects.mp4 • blender-3-edit.mp4 • blender-4-mods.mp4 • blender-5-nurbs.mp4 • blender-6-materials.mp4 • blender-7-animation.mp4 • blender-8-physic+light.mp4 • blender-9-python-ui.mp4	• 1-blender-py-overview.mp4 • 2-blender-py-object.mp4 • 3-blender-py-bezier.mp4 • 4-blender-py-modifier.mp4 • 5-blender-py-material.mp4 • 6-blender-py-skelet.mp4





Пример выполнения комплексного задания



Ссылки на видео

https://www.dropbox.com/sh/oog0rqfk6ot62gn/AAAcFio-g3oQ45CvP8XoflUMa?dl=0?dl=0

Самостоятельная работа

Примерное задание по моделированию

Смоделировать что-то из списка:

- Пингвин
- Снеговик
- Робот
- Пряничный человек
- Медведь
- Придумать любого персонажа, например, состоящего из кубиков или шариков
- Сделать модель по теме работы OpenGL

Для получившейся модели задать материалы (цвет, текстуры и прочее).

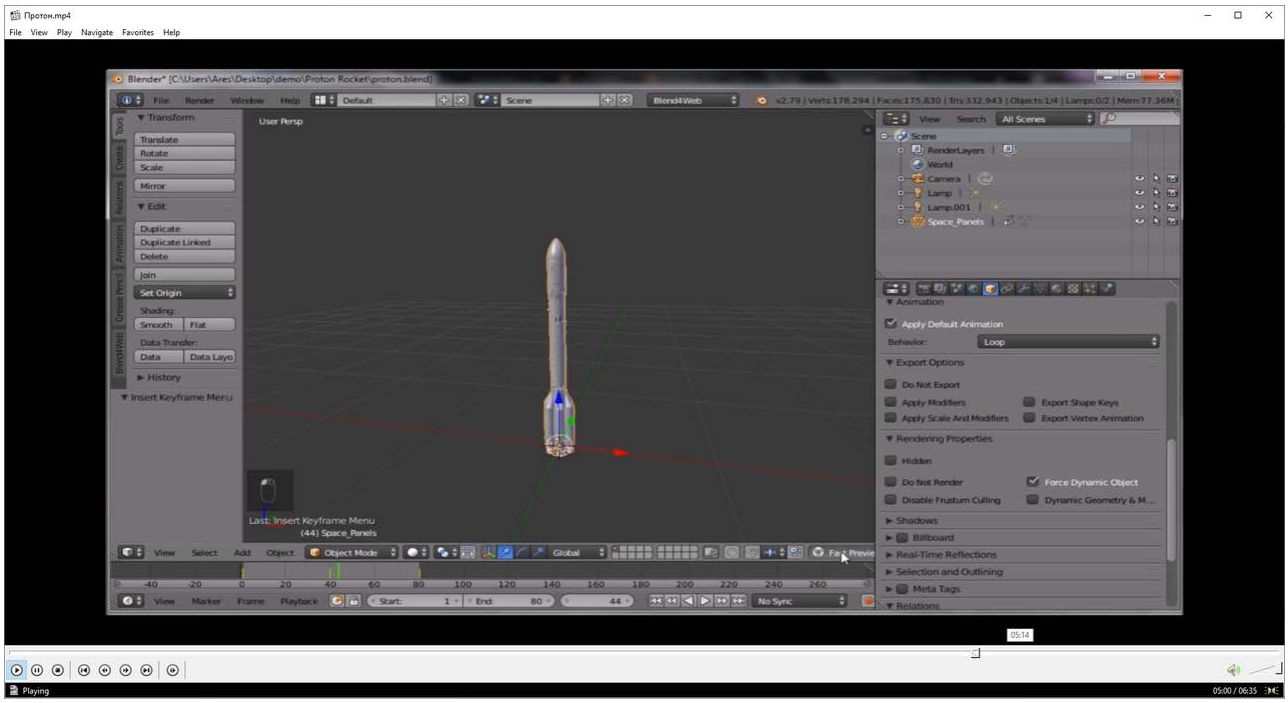
Настроить источники освещения в сцене, попробовать добавить различные типы освещения, для получения оптимального результата.
Сделать простой скелет для модели и создать анимацию, например, походка, взмахи руками, поворот головы и прочее.
Реализовать в сцене физический эффект, например, взаимодействие персонажа с мячиком, сделать ему одежду, волосы, добавить в ветер, воду, дым и т.п.

Примерные задания по скриптам (BPY)

- Создание объекта: пирамида, октаэдр, тетраэдр и т.п.
- Вариант одного из модификаторов Blender, например, Array
- Свой вариант полезного модификатора
- Импорт 3D-модели из файла (например, obj)
- Создание полигональной кривой Безье в виде буквы: А, Б, В и т.п.
- Измерение длин кривых Безье
- Добавление точек в кривые Безье
- Для заданных объектов вычислить их булеву разницу
- Установить pinned-группу для модификатора Cloth
- Сделать заданный объект прозрачным
- Изменить цвет объекта
- Анимация: приседание, вращение головой, взмахи руками, сгибание ног, походка

Задание для Blend4Web/Verge3D

Конвертировать сцену в Blend4Web или лучше Verge3D как в следующем примере.



Литература

1. Прахов А. Самоучитель Blender 2.7. СПб.: БХВ-Петербург, 2016. – 400 с.
2. Blender.org — официальная страница Blender 3D.