Федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский университет ИТМО»

Факультет программной инженерии и компьютерной техники

Алгоритмы компьютерной графики

Отчет по лабораторной работе №4

Выполнил:
Козлов Василий Сергеевич
P3315

Проверил:
Меженин Александр Владимирович

Санкт-Петербург, 8 ноября 2025

Содержание

1 Задание

Исходные данные: Система координат, точечные источники света с Ламбертовской диаграммой излучения, прямоугольный экран, координаты наблюдателя, координаты центра сферы и ее радиус, свойства поверхности сферы (согласно модели Блинн-Фонга).

Цель работы: Овладеть навыками расчета и визуализации распределения яркости на диффузной сфере, освещенной точечными источниками света.

Задачи:

- Провести расчет распределения яркости на сфере в пределах заданной области.

Рекомендуемые пределы значений параметров для расчета:

- Размер прямоугольного экрана по высоте (H) и ширине (W) варьируются в диапазоне от 100 до 10000 миллиметров.
- Разрешение изображения по высоте (Hres) и ширине (Wres) варьируются в диапазоне от 200 до 800 пикселей. Разрешение должно обеспечивать квадратные пиксели.
- Координаты источников света (xLi, yLi, zLi) [мм] по осям X и Y X$\pm$10000, по оси Z от 100 до 10000.
- Координаты наблюдателя: (0, 0, zO) [мм].
- Координаты центра сферы (xC, yC, zC) [мм] по осям X и Y $\pm$10000, по оси Z от 100 до 10000. Сфера должна целиком помещаться в область видимости.
- Сила излучения I0 варьируется от 0.01 до 10000 Вт/ср.
- Параметры модели Блинн-Фонга. Выбираются так, чтобы на изображении был виден ярко выраженный пик.
- Написать приложение на Python, формирующее изображение рассчитанного распределения яркости для заданного разрешения с нормировкой (0-255) на максимальное значение яркости.
- Записать сформированное изображение в файл.
- Визуализировать изображение на мониторе.

2 Описание алгоритма

Алгоритм расчета распределения яркости на сфере основан на модели освещения Блинн-Фонга. Расчет проводится для проекции сферы на прямоугольный экран в системе координат, где экран расположен в плоскости z=0, наблюдатель находится в точке (0, 0, zO), центр сферы — в (xC, yC, zC) с радиусом R. Источники света заданы координатами (xL, yL, zL) и интенсивностью I0 (Вт/ср). Параметры модели: коэффициенты диффузного отражения k_diff, зеркального отражения k_spec и показатель блеска shininess.

2.1 Определение точек на сфере

Для каждого пикселя экрана с координатами (x, y) в миллиметрах (x от -W/2 до W/2, y от -H/2 до H/2, дискретизированные по разрешению Wres x Hres) проверяется принадлежность проекции к сфере в ортографической проекции:

Рассчитывается смещение dx = x - xC, dy = y - yC.

Если $dx^2 + dy^2 > R^2$, то точка не лежит на сфере, яркость равна 0.

Иначе рассчитывается координата z сферы: $z = zC + \sqrt{R^2 - dx^2 - dy^2}$.

Точка на сфере P = (x, y, z).

2.2 Расчет нормали и векторов

Нормаль N в точке P рассчитывается как:

N = $(\frac{x-xC}{R}, \frac{y-yC}{R}, \frac{z-zC}{R})$.

Вектор взгляда V от точки P к наблюдателю: V = (0 - x, 0 - y, zO - z), нормализованный делением на его длину |V|.

Для каждого источника света рассчитывается:

Вектор L от P к источнику: $L = (xL - x, yL - y, zL - z)$, нормализованный $L_{dir} = \frac{L}{|L|}$.

Полувектор $H = \frac{L_{dir}+V}{|L_{dir}+V|}$.

2.3 Расчет компонент яркости по модели Блинн-Фонга

Диффузная компонента: diff = max(0, N · L_dir), где · — скалярное произведение.

Зеркальная компонента: $spec = max(0, N \cdot H)^{shininess}$.

Яркость от одного источника: $I_0 \cdot (k_{diff} \cdot diff + k_{spec} \cdot spec)$.

Общая яркость I в точке — сумма по всем источникам.

Если точка не на сфере, I = 0.

2.4 Формирование изображения

Массив яркостей нормализуется: значение пикселя = $\frac{I}{I_{max}} \cdot 255$, где I_max — максимальная яркость на сфере. Результат преобразуется в целочисленный тип от 0 до 255 для градаций серого.

Изображение сохраняется в файл формата PNG.

2.5 Расчет яркости в заданных точках

Для заданных точек на экране (x, y) в мм рассчитывается соответствующий индекс пикселя:

u = round(((x + W/2) / W) * (Wres - 1))

v = round(((H/2 - y) / H) * (Hres - 1))

Если индекс в пределах и яркость > 0, возвращается абсолютное значение I, иначе — сообщение об ошибке.

3 Результаты

Расчет проведен для следующих параметров:

- H = 500 мм, W = 500 мм
- Hres = 360 пикс, Wres = 360 пикс
- I0 = 10 Вт/ср
- k_diff = 0.8, k_spec = 0.5, shininess = 20
- Источник света: (2000, 0, 2000) мм
- Центр сферы: (0, 0, 1000) мм, радиус R = 300 мм
- Положение наблюдателя: zO = 1500 мм

Максимальная яркость на сфере: 11.7475 Вт/ср
Минимальная яркость на сфере (ненулевая): 1e-45 Вт/ср (учитывая числовую точность, минимальное значимое значение близко к нулю на границах)
Расчетные значения яркости в трех точках на экране (соответствующих точкам на сфере):

- Точка P1 (0.0, 0.0) мм: 2.7488 Вт/ср
- Точка P2 (150.0, 0.0) мм: 11.0240 Вт/ср
- Точка P3 (-150.0, 0.0) мм: 1.1835e-07 Вт/ср

Результирующее изображение (нормализованное распределение яркости):

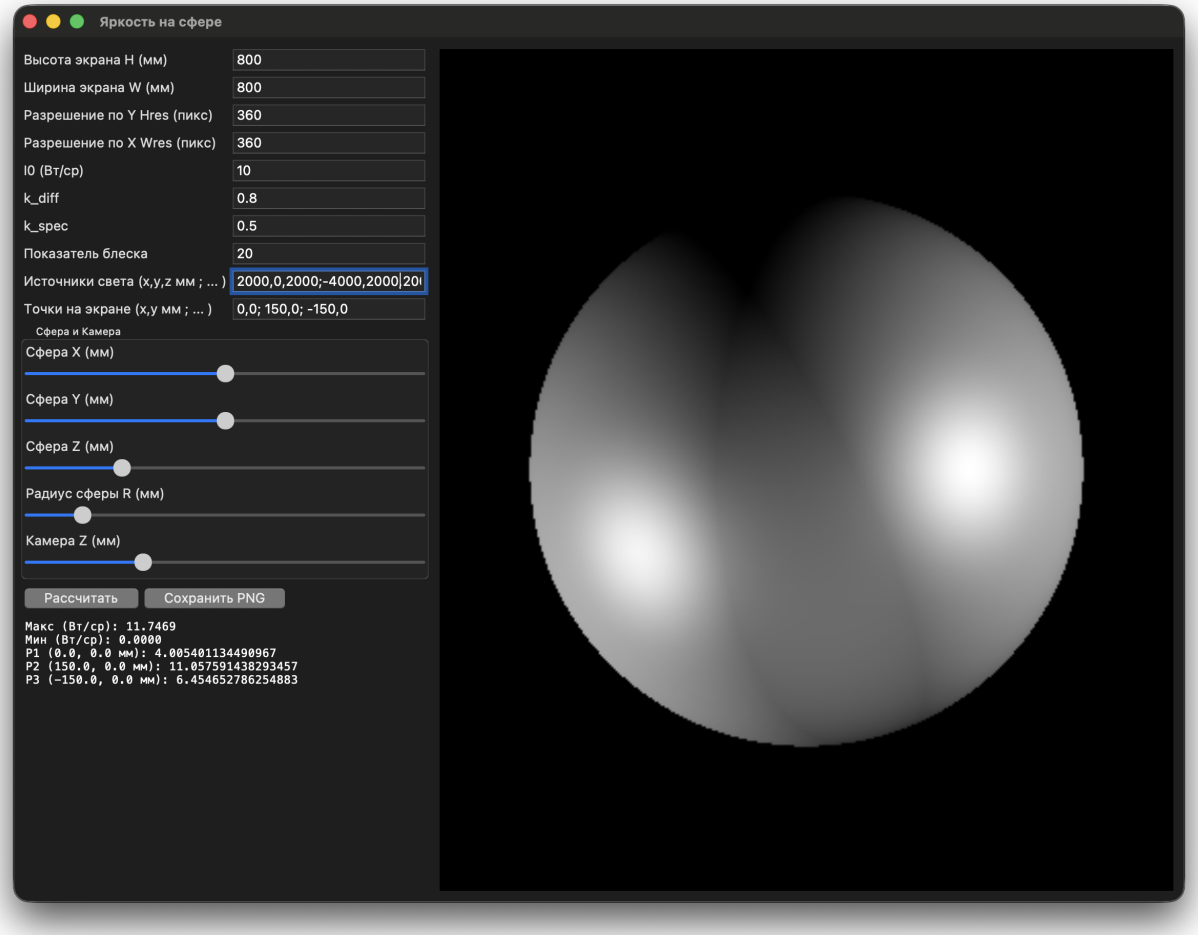

Рис. 1: Пример расчета в программе

4 Исходный код программы

4.1 calc.py

```
import numpy as np

def ray_sphere_intersection_np(ray_origin, ray_dir, center, radius):
    oc = ray_origin - center
    b = np.sum(ray_dir * oc, axis=-1)
    c = np.sum(oc * oc, axis=-1) - radius * radius
    h = b * b - c
    hit = h >= 0.0
    t = -b - np.sqrt(np.maximum(h, 0.0))
    return hit, t


def compute_brightness(params):
    H, W = params["H"], params["W"]
    Hres, Wres = params["Hres"], params["Wres"]

    xC, yC, zC = params["sphere_center"]
    R = params["sphere_radius"]
    zO = params["observer_z"]
    I0 = params["I0"]
    k_diff = params["k_diff"]
    k_spec = params["k_spec"]
    shininess = params["shininess"]
    lights = params["light_sources"]
    perspective = params["perspective"]

    xs = np.linspace(-W / 2.0, W / 2.0, Wres)
    ys = np.linspace(-H / 2.0, H / 2.0, Hres)
    xv, yv = np.meshgrid(xs, ys, indexing="xy")

    dx = xv - xC
    dy = yv - yC
    inside = dx * dx + dy * dy <= R * R
    img = np.zeros((Hres, Wres), dtype=np.float32)

    # если все за пределами или источников нет возвращаем сразу нули
    if not np.any(inside) or len(lights) == 0:
        return img

    # координата z на сфере
    z_sphere = zC + np.sqrt(np.maximum(R * R - dx * dx - dy * dy, 0.0))

    # вектор нормали
    nx = (xv - xC) / R
    ny = (yv - yC) / R
    nz = (z_sphere - zC) / R
```

```
    # формируем вектора нормалей
    N = np.stack([nx, ny, nz], axis=-1)[..., None, :]

    # вектор взгляда
    V = np.stack([np.zeros_like(xv), np.zeros_like(yv), zO - z_sphere], axis=
    V /= np.linalg.norm(V, axis=-1, keepdims=True)

    # массив векторов вида (x, y, z) с координатами источников света
    lights_arr = np.array(lights, dtype=np.float32)

    # формируем вектор от точки P (точка на сфере) к источнику
    Lvec = lights_arr[None, None, :, :] - np.stack(
        [xv[..., None], yv[..., None], z_sphere[..., None]], axis=-1
    )

    Lnorm = np.linalg.norm(Lvec, axis=-1, keepdims=True)

    # избегаем деления на 0
    Ldir = Lvec / np.maximum(Lnorm, 1e-9)

    # рассчитываем диффузию в каждой точке от каждого источника
    # векторное произведение представили в виде суммы произведения компонент
    diff = np.maximum(np.sum(N * Ldir, axis=-1), 0.0)

    Vtile = V[..., None, :]

    # рассчитываем полувектор
    Hvec = Ldir + Vtile
    Hvec /= np.maximum(np.linalg.norm(Hvec, axis=-1, keepdims=True), 1e-9)

    # рассчитываем спектральную составляющую в каждой точке от каждого
источника
    # векторное произведение представили в виде суммы произведения компонент
    spec = np.maximum(np.sum(N * Hvec, axis=-1), 0.0) ** shininess

    # для каждого источника вычисляем яркость и суммируем
    # яркость считается по модели Блинна–Фонга
    total = np.sum(I0 * (k_diff * diff + k_spec * spec), axis=-1)
    img += total
    img[~inside] = 0.0

    return img
```

4.2 application.py

```
#!/usr/bin/env python3

import sys
import tkinter as tk
```

```
from tkinter import ttk, filedialog, messagebox
from PIL import Image, ImageTk
import numpy as np
from calc import *

class BrightnessApp:
    def __init__(self, root):
        self.root = root
        root.title("Яркость на сфере")

        main_frame = ttk.Frame(root)
        main_frame.pack(fill="both", expand=True, padx=4, pady=4)

        self.frame = ttk.Frame(main_frame)
        self.frame.pack(side="left", fill="y", padx=4, pady=4)

        self.fields = {}
        self._add_field("H", "Высота экрана H (мм)", 500)
        self._add_field("W", "Ширина экрана W (мм)", 500)
        self._add_field("Hres", "Разрешение по Y Hres (пикс)", 360)
        self._add_field("Wres", "Разрешение по X Wres (пикс)", 360)
        self._add_field("I0", "I0 (Вт/ср)", 10)
        self._add_field("k_diff", "k_diff", 0.8)
        self._add_field("k_spec", "k_spec", 0.5)
        self._add_field("shininess", "Показатель блеска", 20)

        ttk.Label(self.frame, text="Источники света (x,y,z мм; ...)").grid(
            row=len(self.fields), column=0, sticky="w"
        )

        self.lights_entry = ttk.Entry(self.frame)
        self.lights_entry.insert(0, "2000,0,2000")
        self.lights_entry.grid(row=len(self.fields), column=1, sticky="ew")
        self.frame.columnconfigure(1, weight=1)

        points_row = len(self.fields) + 1
        ttk.Label(self.frame, text="Точки на экране (x,y мм; ...)").grid(
            row=points_row, column=0, sticky="w"
        )
        self.points_entry = ttk.Entry(self.frame)
        self.points_entry.insert(0, "0,0; 150,0; -150,0")
        self.points_entry.grid(row=points_row, column=1, sticky="ew")

        check_row = points_row + 1
        self.perspective_var = tk.BooleanVar(value=False)
        # ttk.Checkbutton(
        #     self.frame,
        #     text="Перспектива",
        #     variable=self.perspective_var,
        #     command=self.compute,
```

```
        # ).grid(row=check_row, column=0, columnspan=2, sticky="w")

        slider_row = check_row + 1
        slider_frame = ttk.LabelFrame(self.frame, text="Сфера и Камера")
        slider_frame.grid(row=slider_row, column=0, columnspan=2, pady=2, sti

        self.sph_x = self._add_slider(slider_frame, "Сфера X (мм)", -2000, 20
        self.sph_y = self._add_slider(slider_frame, "Сфера Y (мм)", -2000, 20
        self.sph_z = self._add_slider(slider_frame, "Сфера Z (мм)", 100, 4000
        self.sph_r = self._add_slider(
            slider_frame, "Радиус сферы R (мм)", 50, 2000, 300
        )

        self.cam_z = self._add_slider(slider_frame, "Камера Z (мм)", 100, 500

        btn_row = slider_row + 1
        btn_frame = ttk.Frame(self.frame)
        btn_frame.grid(row=btn_row, column=0, columnspan=2, pady=2, sticky="w
        ttk.Button(btn_frame, text="Рассчитать", command=self.compute).pack(
            side="left", padx=2
        )
        ttk.Button(btn_frame, text="Сохранить PNG", command=self.save_png).pa
            side="left", padx=2
        )

        results_row = btn_row + 1
        self.results_text = tk.Text(self.frame, height=6, width=40, wrap="wor
        self.results_text.grid(
            row=results_row, column=0, columnspan=2, pady=2, sticky="w"
        )

        self.canvas = tk.Canvas(main_frame, bg="black")
        self.canvas.pack(side="right", fill="both", expand=True, padx=4, pady

        self.current_image = None
        self.preview_tk = None
        self.arr = None
        self.params = None

        self.root.bind("<Configure>", self.on_resize)
        self._debounce_id = None
        self.root.after(150, self.auto_update)

    def _add_field(self, key, label, default):
        row = len(self.fields)
        ttk.Label(self.frame, text=label).grid(row=row, column=0, sticky="w")
        entry = ttk.Entry(self.frame, width=10)
        entry.insert(0, str(default))
        entry.grid(row=row, column=1, sticky="ew")
        self.fields[key] = entry
```

```
def _add_slider(self, parent, text, mn, mx, val):
    container = ttk.Frame(parent)
    ttk.Label(container, text=text).pack(anchor="w")
    var = tk.DoubleVar(value=val)
    scale = ttk.Scale(
        container,
        from_=mn,
        to=mx,
        orient="horizontal",
        variable=var,
        command=lambda v: self._debounced_compute(),
    )
    scale.pack(fill="x")
    container.pack(fill="x")
    return var

def parse_lights(self):
    raw = self.lights_entry.get().strip()
    if not raw:
        return []
    parts = [p.strip() for p in raw.split(";") if p.strip()]
    lights = []
    for p in parts:
        try:
            x, y, z = map(float, p.split(","))
            lights.append((x, y, z))
        except Exception:
            continue
    return lights

def parse_points(self):
    raw = self.points_entry.get().strip()
    if not raw:
        return []
    parts = [p.strip() for p in raw.split(";") if p.strip()]
    points = []
    for p in parts:
        try:
            x, y = map(float, p.split(","))
            points.append((x, y))
        except Exception:
            continue
    return points

def auto_update(self):
    self.compute()
    self.root.after(300, self.auto_update)

def _debounced_compute(self, delay=80):
```

```
    if self._debounce_id:
        try:
            self.root.after_cancel(self._debounce_id)
        except Exception:
            pass
    self._debounce_id = self.root.after(delay, self.compute)

def compute(self):
    try:
        p = {k: float(e.get()) for k, e in self.fields.items()}
        params = {
            "H": p["H"],
            "W": p["W"],
            "Hres": int(max(64, min(1024, int(p["Hres"])))),
            "Wres": int(max(64, min(1024, int(p["Wres"])))),
            "sphere_center": (
                float(self.sph_x.get()),
                float(self.sph_y.get()),
                float(self.sph_z.get()),
            ),
            "sphere_radius": float(self.sph_r.get()),
            "observer_z": float(self.cam_z.get()),
            "I0": p["I0"],
            "k_diff": p["k_diff"],
            "k_spec": p["k_spec"],
            "shininess": p["shininess"],
            "light_sources": self.parse_lights(),
            "perspective": bool(self.perspective_var.get()),
        }

        arr = compute_brightness(params)
        self.arr = arr
        self.params = params

        maxv = float(np.max(arr)) if arr.size else 0.0

        if maxv > 0:
            img_u8 = (arr / maxv * 255.0).clip(0, 255).astype(np.uint8)
        else:
            img_u8 = arr.astype(np.uint8)

        self.current_image = Image.fromarray(
            img_u8, mode="L"
        )

        self.display_image()

        minv = np.min(arr[arr > 0]) if np.any(arr > 0) else 0.0
        points = self.parse_points()
        brights = []
```

```
            for pt in points:
                x, y = pt
                u = ((x + params["W"] / 2.0) / params["W"]) * (params["Wres"]
                v = ((params["H"] / 2.0 - y) / params["H"]) * (params["Hres"]
                u_int = int(round(u))
                v_int = int(round(v))
                if (
                    0 <= u_int < params["Wres"]
                    and 0 <= v_int < params["Hres"]
                    and arr[v_int, u_int] > 0
                ):
                    brights.append(arr[v_int, u_int])
                else:
                    brights.append("Неверная точка (за пределами или не на сф

            self.results_text.delete("1.0", tk.END)

            info = f"Макс (Вт/ср): {maxv:.4f}\n"
            info += f"Мин (Вт/ср): {minv:.4f}\n"

            for i in range(min(3, len(brights))):
                pt_str = ", ".join(f"{c:.1f}" for c in points[i])

                if isinstance(brights[i], float):
                    info += f"P{i+1} ({pt_str} мм): {brights[i]:.4f} (Вт/ср)\
                else:
                    info += f"P{i+1} ({pt_str} мм): {brights[i]}\n"

            self.results_text.insert(tk.END, info)

        except Exception as e:
            print("Ошибка расчёта:", e, file=sys.stderr)

    def display_image(self):
        if self.current_image is None:
            return

        w = self.canvas.winfo_width()
        h = self.canvas.winfo_height()

        if w <= 1 or h <= 1:
            return

        aspect = self.current_image.width / self.current_image.height

        if w / h > aspect:
            new_h = h
            new_w = int(h * aspect)
        else:
```

```
            new_w = w
            new_h = int(w / aspect)

        preview = self.current_image.resize((new_w, new_h), resample=Image.BI
        self.preview_tk = ImageTk.PhotoImage(preview)
        self.canvas.delete("all")
        self.canvas.create_image(
            (w - new_w) // 2, (h - new_h) // 2, anchor="nw", image=self.previ
        )

    def on_resize(self, event):
        self.display_image()

    def save_png(self):
        if self.current_image is None:
            messagebox.showinfo(
                "Нет изображения", "Пожалуйста, рассчитайте изображение снача
            )
            return
        path = filedialog.asksaveasfilename(
            defaultextension=".png", filetypes=[("PNG изображение", "*.png")]
        )
        if not path:
            return
        try:
            self.current_image.save(path, format="PNG")
            messagebox.showinfo("Сохранено", f"Изображение сохранено в:\n{pat
        except Exception as e:
            messagebox.showerror("Ошибка сохранения", str(e))


if __name__ == "__main__":
    root = tk.Tk()
    app = BrightnessApp(root)
    root.mainloop()
```

5 Выводы

В ходе выполнения лабораторной работы был разработан алгоритм расчета распределения яркости на сфере, освещенной точечными источниками света, на основе модели Блинн-Фонга. Алгоритм позволяет определять точки на сфере, рассчитывать нормали и вектора, а также компоненты диффузного и зеркального отражения без учета затухания с расстоянием.

Полученные результаты демонстрируют градиент яркости от максимума в области, близкой к источнику света и блику, до минимума на противоположной стороне. Максимум яркости соответствует области с сильным зеркальным отражением, а минимум — затененным участкам.

Работа позволила овладеть принципами расчета освещения в компьютерной графике, пониманием модели Блинн-Фонга и методами проекции объектов на экран.