**Министерство науки и высшего образования российской федерации**

**Федеральное государственное автономное образовательное учреждение высшего образования**

**“НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО”**

|  |  |
| --- | --- |
| **Факультет** | **Программной Инженерии и Компьютерной Техники** |
| **Направление подготовки (специальность)** | **Компьютерные технологии в дизайне** |
| **Дисциплина** | **Алгоритмы компьютерной графики** |

Лабораторная работа 2

ОТЧЕТ

|  |  |
| --- | --- |
| **Выполнил студент:** | **Хоанг Ван Куан (345124)** |
| **Группа:** | **P3468** |
| **Преподаватель:** | **Меженин Александр Владимирович** |

г. Санкт-Петербург

2025

# ОТЧЕТ О ХОДЕ ВЫПОЛНЕНИЯ

# 

### Цель работы

Изучить методы обработки изображений на клиентской стороне с использованием JavaScript и Canvas API, а также построение гистограмм для визуализации распределения цветовых каналов (R, G, B).

### Описание реализации

* Canvas-элементы для отображения исходного изображения и вариантов (яркость, контраст, инверсия).
* Canvas-элементы для гистограмм каждого изображения.
* Ползунки <input type="range"> для управления значениями фильтров.
* Кнопка или поле для загрузки изображения

### Основные функции

1. uploadImage(image)

Загружает изображение в память.

Если передан объект File, используется FileReader для преобразования в Data URL.

После загрузки изображения вызываются функции:

* loadImage – для отображения изображения на canvas.
* calculate – для применения фильтров (яркость, контраст, инверсия).
* drawHistValueLabel – для отображения текстовых меток гистограмм.
* async function uploadImage(image) {
* let img = new Image();
* if (image instanceof File) {
* const url = await readAsDataURL(image);
* img.src = url;
* } else img.src = image;
* await img.decode();
* imageGlobal.src = img.src;
* loadImage(img, orig, histOrig);
* calculate(orig, bright, histBright, brightSlider.value / 100, "brightness");
* calculate(orig, cont, histCont, contrastSlider.value, "contrast");
* calculate(orig, inverted, histInverted, invertedSlider.value / 100, "inverted");
* }

1. drawImageProp(ctx, img, x, y, w, h, offsetX, offsetY)

Корректное отображение изображения на canvas с сохранением пропорций.

Позволяет выбрать часть изображения для отображения с помощью offsetX и offsetY.

Изображение масштабируется и центрируется на canvas.

function drawImageProp(ctx, img, x, y, w, h, offsetX, offsetY) {

  if (arguments.length === 2) {

    x = y = 0;

    w = ctx.canvas.width;

    h = ctx.canvas.height;

  }

  offsetX = typeof offsetX === "number" ? offsetX : 0.5;

  offsetY = typeof offsetY === "number" ? offsetY : 0.5;

  if (offsetX < 0) offsetX = 0;

  if (offsetY < 0) offsetY = 0;

  if (offsetX > 1) offsetX = 1;

  if (offsetY > 1) offsetY = 1;

  var iw = img.width,

    ih = img.height,

    r = Math.min(w / iw, h / ih),

    nw = iw \* r,

    nh = ih \* r,

    ar = 1;

  if (nw < w) ar = w / nw;

  if (Math.abs(ar - 1) < 1e-14 && nh < h) ar = h / nh;

  nw \*= ar;

  nh \*= ar;

  var cw = iw / (nw / w);

  var ch = ih / (nh / h);

  var cx = (iw - cw) \* offsetX;

  var cy = (ih - ch) \* offsetY;

  if (cx < 0) cx = 0;

  if (cy < 0) cy = 0;

  if (cw > iw) cw = iw;

  if (ch > ih) ch = ih;

  ctx.drawImage(img, cx, cy, cw, ch, x, y, w, h);

}

1. calculate(origCanvas, canvas, histCanvas, value, type)

Получает данные пикселей исходного изображения (ImageData).

В зависимости от type применяет:

Контраст: newValue = (oldValue - 128) \* factor + 128.

Яркость: newValue = oldValue \* value.

Инверсия: newValue = 255 - oldValue \* value.

После обработки обновляет canvas с изображением и строит гистограмму.

async function calculate(origCanvas, canvas, histCanvas, value, type) {

  let ctx = canvas.getContext("2d");

  let ctxOrig = origCanvas.getContext("2d");

  const w = origCanvas.width;

  const h = origCanvas.height;

  if (w === 0 || h === 0) return;

  // попиксельные данные

  let imageData = ctxOrig.getImageData(0, 0, w, h);

  let data = imageData.data;

  if (type === "contrast") {

    let contrast = value \* (255 / 100);

    let factor = (255 + contrast) / (255.01 - contrast);

    for (let i = 0; i < data.length; i += 4) {

      data[i] = clamp(Math.round(factor \* (data[i] - 128) + 128));

      data[i + 1] = clamp(Math.round(factor \* (data[i + 1] - 128) + 128));

      data[i + 2] = clamp(Math.round(factor \* (data[i + 2] - 128) + 128));

    }

  } else if (type === "brightness") {

    for (let i = 0; i < data.length; i += 4) {

      data[i] = clamp(Math.round(data[i] \* value));

      data[i + 1] = clamp(Math.round(data[i + 1] \* value));

      data[i + 2] = clamp(Math.round(data[i + 2] \* value));

    }

  } else if (type === "inverted") {

    for (let i = 0; i < data.length; i += 4) {

      data[i] = clamp(Math.round(255 - data[i] \* value));

      data[i + 1] = clamp(Math.round(255 - data[i + 1] \* value));

      data[i + 2] = clamp(Math.round(255 - data[i + 2] \* value));

    }

  }

  ctx.putImageData(imageData, 0, 0);

  processImage(ctx.getImageData(0, 0, canvas.width, canvas.height).data, histCanvas, type);

}

1. processImage(data, histCanvas)

Строит гистограмму для каждого цветового канала (R, G, B).

Нормализует высоту колонок по максимальному значению яркости.

Добавляет подписи цветовых каналов и отображает их цвет.

function processImage(data, histCanvas, type) {

  let histR = new Array(256).fill(0);

  let histG = new Array(256).fill(0);

  let histB = new Array(256).fill(0);

  for (let i = 0; i < data.length; i += 4) {

    histR[data[i]]++;

    histG[data[i + 1]]++;

    histB[data[i + 2]]++;

  }

  let maxBrightness = Math.max(...histR, ...histG, ...histB, 1);

  const ctx = histCanvas.getContext("2d");

  let dx = histCanvas.width / 256;

  let dy = histCanvas.height / maxBrightness;

  ctx.fillStyle = "#fff";

  ctx.fillRect(0, 0, histCanvas.width, histCanvas.height);

  ctx.lineWidth = Math.max(1, Math.floor(dx));

  for (let i = 0; i < 256; i++) {

    let x = i \* dx + dx / 2;

    ctx.strokeStyle = "rgba(255,0,0,0.55)";

    ctx.beginPath();

    ctx.moveTo(x, histCanvas.height);

    ctx.lineTo(x, histCanvas.height - histR[i] \* dy);

    ctx.stroke();

    ctx.strokeStyle = "rgba(0,255,0,0.55)";

    ctx.beginPath();

    ctx.moveTo(x, histCanvas.height);

    ctx.lineTo(x, histCanvas.height - histG[i] \* dy);

    ctx.stroke();

    ctx.strokeStyle = "rgba(0,0,255,0.55)";

    ctx.beginPath();

    ctx.moveTo(x, histCanvas.height);

    ctx.lineTo(x, histCanvas.height - histB[i] \* dy);

    ctx.stroke();

  }

  ctx.font = "12px Arial";

  ctx.textBaseline = "top";

  const padding = 6;

  const gap = 28;

  ctx.fillStyle = "red";

  ctx.fillRect(padding, padding, 12, 12);

  ctx.fillStyle = "#000";

  ctx.fillText("R", padding + 16, padding);

  ctx.fillStyle = "green";

  ctx.fillRect(padding + gap, padding, 12, 12);

  ctx.fillStyle = "#000";

  ctx.fillText("G", padding + gap + 16, padding);

  ctx.fillStyle = "blue";

  ctx.fillRect(padding + gap \* 2, padding, 12, 12);

  ctx.fillStyle = "#000";

  ctx.fillText("B", padding + gap \* 2 + 16, padding);

  // label

  let textLabel = "Original";

  if (type === "brightness") textLabel = `Brightness: ${safeInt(brightSlider.value)}%`;

  if (type === "contrast") textLabel = `Contrast: ${safeInt(contrastSlider.value)}`;

  if (type === "inverted") textLabel = `Inverted: ${safeInt(invertedSlider.value)}%`;

  ctx.save();

  ctx.font = "12px Arial";

  ctx.textBaseline = "top";

  const metrics = ctx.measureText(textLabel);

  const textW = metrics.width;

  const textH = 14;

  const x = histCanvas.width - textW - padding;

  const y = padding;

  ctx.fillStyle = "rgba(255,255,255,0.8)";

  ctx.fillRect(x - 4, y - 2, textW + 8, textH + 4);

  ctx.fillStyle = "#000";

  ctx.fillText(textLabel, x, y);

  ctx.restore();

}

1. drawHistValueLabel(histCanvas, label)

Отображает текстовую метку значения фильтра в правом верхнем углу canvas с полупрозрачным фоном.

Используется для наглядного отображения текущих параметров изображения.

function drawImageProp(ctx, img, x, y, w, h, offsetX, offsetY) {

  if (arguments.length === 2) {

    x = y = 0;

    w = ctx.canvas.width;

    h = ctx.canvas.height;

  }

  offsetX = typeof offsetX === "number" ? offsetX : 0.5;

  offsetY = typeof offsetY === "number" ? offsetY : 0.5;

  if (offsetX < 0) offsetX = 0;

  if (offsetY < 0) offsetY = 0;

  if (offsetX > 1) offsetX = 1;

  if (offsetY > 1) offsetY = 1;

  var iw = img.width,

    ih = img.height,

    r = Math.min(w / iw, h / ih),

    nw = iw \* r,

    nh = ih \* r,

    ar = 1;

  if (nw < w) ar = w / nw;

  if (Math.abs(ar - 1) < 1e-14 && nh < h) ar = h / nh;

  nw \*= ar;

  nh \*= ar;

  var cw = iw / (nw / w);

  var ch = ih / (nh / h);

  var cx = (iw - cw) \* offsetX;

  var cy = (ih - ch) \* offsetY;

  if (cx < 0) cx = 0;

  if (cy < 0) cy = 0;

  if (cw > iw) cw = iw;

  if (ch > ih) ch = ih;

  ctx.drawImage(img, cx, cy, cw, ch, x, y, w, h);

}

# Результаты

# 

# 