Dr. Ole Klein, Dr. Steffen Müthing
IWR, Universität Heidelberg

Abgabe 15. Dezember 2017

Übung 1 *Versionskontrolle mit git*

git ist, wie in der Vorlesung vorgestellt, ein verteiltes Versionskontrollsystem. In dieser Übung geht es darum, sich ein wenig mit dem System vertraut zu machen.

(a) Melden Sie sich bei einem der Dienste GitLab[1], Bitbucket[2] oder GitHub[3] an - da GitLab und BitBucket auch private Repositories erlauben, sind sie etwas besser geeignet.

(b) Erstellen Sie auf der Website ein neues Projekt für die Übungen, z.B. `ipk-exercises`, und folgen Sie den Anweisungen auf der Website, um dieses Repository auf Ihren Computer zu klonen.
Wichtig: Auf GitLab folgen Sie den Schritten nur bis zum Ende des Absatzes "Create a new repository"!

(c) Kopieren Sie die Quellcodedateien der bisherigen übungen in das neue Verzeichnis, fügen Sie sie zu git hinzu und erstellen einen Commit. Beim Erstellen des Commits müssen Sie eine Commit Message eingeben. Dies können Sie auf zwei verschiedene Arten machen:

- Sie schreiben einfach `git commit`. In diesem Fall öffnet sich im Terminal ein Texteditor mit einigen auskommentierten Zeilen (beginnend mit `"#"`). Schreiben Sie in die erste (leere) Zeile die Commit Message (diese kann auch mehrere Zeilen lang sein), speichern Sie die Datei und verlassen Sie den Editor. Meistens öffnet sich unter Linux der Editor `vim`, der etwas gewöhnungsbedürftig ist:
  (a) Drücken Sie die Taste `"i"` (für "insert"), um den Eingabemodus zu aktivieren.
  (b) Tippen Sie die commit message.
  (c) Drücken Sie die Taste `"ESC"`, um den Editiermodus zu verlassen.
  (d) Geben Sie folgendes ein, um die Datei zu speichern und den Editor zu schliessen: `":wq"` und drücken Sie Enter.
- Sie können die commit message auch auf der Kommandozeile eingeben:

```
1 git commit -m "hier steht Ihre commit message"
```

  Damit ist es allerdings schwierig, mehrzeilige Commit Messages zu schreiben.

(d) Pushen Sie den Commit auf den Server.

(e) Während der Arbeit in den Übungen entstehen diverse Dateien, die nicht mit unter Versionskontrolle sollen, z.B. Sicherungsdateien, `.o`-Dateien und CMake-Buildverzeichnisse. Um diese Dateien zu *ignorieren*, gibt es in git den `gitignore`-Mechanismus (siehe `git help gitignore`). Erstellen Sie in Ihrem Arbeitsverzeichnis eine Datei `.gitignore` (Achtung, diese Datei ist versteckt, zum Anzeigen verwenden Sie `ls -a`) mit Dateinamen, die nicht zu git hinzugefügt werden sollen, z.B.

---

[1] `https://gitlab.org`
[2] `https://bitbucket.org`
[3] `https://github.com`

```
# alle Dateien, die auf .o enden
*.o
# eine bestimmte Datei, z.B. ein Executable
statistics
# Sicherungskopien, die auf ~ enden
*~
# das Unterverzeichnis build/ (von CMake)
build/
```

Fügen Sie die Datei zu git hinzu, erstellen Sie einen Commit und pushen Sie diesen.

Von nun an können Sie git verwenden, um Dateien zwischen Ihrem Laptop und dem Rechner im Computer-Pool zu synchronisieren.

Die nächste Aufgabe auf dem Übungsblatt dürfen Sie in Zweierteams bearbeiten. Hierfür ist es sinnvoll, wenn beide von Ihnen auf dasselbe Server-Repository zugreifen können.

(f) Suchen Sie sich einen Partner, mit dem Sie die nächste Aufgabe zusammen bearbeiten wollen.

(g) Wählen Sie ein Server-Projekt, das Sie zusammen verwenden möchten. Sie können entweder das normale Übungs-Projekt von einem von Ihnen verwenden oder einen von Ihnen legt ein neues Projekt an.

(h) Jetzt müssen Sie dem anderen noch Zugriff geben. Auf GitLab funktioniert das Sie auf "Settings → Members" klicken, dort den Benutzernamen eintragen und (**wichtig!**) als role permission "master" eintragen. Damit haben Sie beide volle Zugriffsrechte auf das Projekt.

Weitere Hilfe zu git finden Sie unter anderem hier:

- `https://git-scm.com/doc` Ausführliche Dokumentation und einige Einführungsvideos.
- `https://git-scm.com/docs/everyday` Eine praktische Kurzübersicht wichtiger Befehle.
- `https://git-scm.com/docs/gittutorial` Das offizielle Tutorial.

Ansonsten ist Google euer Freund…

(Grundlagen + Fortgeschritten)

Übung 2 *Pixelgraphiken*

Graphiken werden in Computern auf zwei grundlegend verschiedene Weisen behandelt: mit Vektorgraphiken, die Bilder abstrakt durch Kurvensegmente und geometrische Figuren darstellen, und mit Pixel- bzw. Rastergraphiken, die Bilder als ein diskretes zweidimensionales Gitter aus kleinen Bildpunkten, sogenannten Pixeln, behandeln. In dieser Aufgabe wollen wir uns mit dem letztgenannten Ansatz näher beschäftigen.

Hinweise:

- Die folgende Aufgabe dürfen Sie als Team zu zweit bearbeiten!
- Erstellen Sie für die Aufgabe ein CMake-Buildsystem, das die einzelnen Programme baut.
- Verwenden Sie in allen Konstruktoren eine constructor initializer list.

(a) Schreiben Sie einen Header `point.hh` und eine Implementierung `point.cc` mit einer Klasse namens `Point`, die die Koordinaten eines Punktes in der Ebene speichern kann (zwei **`double`**-Werte). Hierbei ist folgendes zu beachten:

- Die Member-Variablen sollen **`private`** sein.
- Es soll einen Default-Konstruktor geben, der den Punkt $(0, 0)$ erzeugt.

- Es soll einen Konstruktor geben, der die x- und y-Koordinaten als zwei Parameter übergeben bekommt.
- Die Klasse soll Accessor-Methoden

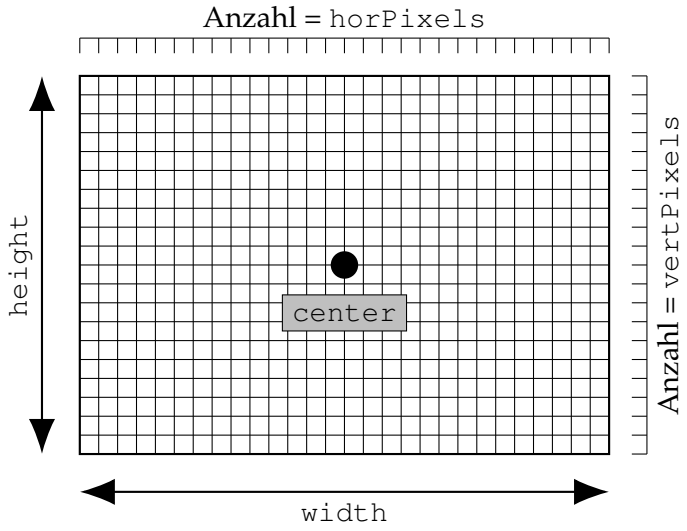
```
double x() const;
double y() const;
```

bereitstellen.

(b) Schreiben Sie einen Header `canvas.hh` und eine Implementierung `canvas.cc` mit einer Klasse `Canvas` (Leinwand), die die Pixel eines Bildes mit zugehöriger Geometrie-Information darstellt. Dies stellen wir durch folgende Informationen dar:

- Eine Anzahl an Pixeln in x- und in y-Richtung.
- Einen `Point`, der die Koordinate der Bildmitte angibt.
- Zwei **`double`**-Werten für die physikalische Höhe und Breite des Bildes.

Das ist auf folgendem Bild noch einmal illustriert:



Dazu soll die Klasse ein zweidimensionales Array zum Speichern von Grauwerten enthalten. Dieses implementieren Sie als `std::vector<std::vector<int> >`. Bis auf das Pixel-Array sollen die Einträge **`const`** sein, damit sie vor unbeabsichtigten Änderungen geschützt sind.

(c) Einen Konstruktor

```
Canvas(const Point& center, double width, double height,
       int horPixels, int vertPixels);
```

der mit `center` als Zentrum ein `Canvas` mit `horPixels` pro Zeile und `vertPixels` pro Spalte konstruiert.

(d) Eine Methode `Point coord(int i, int j) const` von `Canvas`, die mithilfe der gespeicherten Geometrie-Informationen den Punkt berechnet, der dem Pixel in der i-ten Spalte und j-ten Zeile (von links unten aus gesehen) entspricht. Dabei dürfen Sie der Einfachheit halber die linke untere Ecke des Pixels als Punkt ausgeben.

(e) Testen Sie die obige Funktionalität ausgiebig, bevor Sie den Rest der Aufgabe bearbeiten. Gerade beim Berechnen der Koordinaten können leicht Fehler unterlaufen, die man ohne Testen nicht unbedingt sofort bemerkt. Prüfen Sie zumindest die korrekte Berechnung der vier Ecken, die bis auf die Breite eines Pixels mit folgendem übereinstimmen sollten:

- Linke untere Ecke: $i = 0$, $j = 0$, $x = x_{\text{center}} - \frac{1}{2}\text{width}$, $y = y_{\text{center}} - \frac{1}{2}\text{height}$
- Rechte untere Ecke: $i = \text{horPixels} - 1$, $j = 0$, $x = x_{\text{center}} + \frac{1}{2}\text{width}$, $y = y_{\text{center}} - \frac{1}{2}\text{height}$

- Linke obere Ecke: $i = 0$, $j = \text{vertPixels} - 1$, $x = x_{\text{center}} - \frac{1}{2}\text{width}$, $y = y_{\text{center}} + \frac{1}{2}\text{height}$
- Rechte obere Ecke: $i = \text{horPixels} - 1$, $j = \text{vertPixels} - 1$, $x = x_{\text{center}} + \frac{1}{2}\text{width}$, $y = y_{\text{center}} + \frac{1}{2}\text{height}$

(f) Nachdem Sie sichergestellt haben, dass Ihre Methode die korrekten Koordinaten liefert, schreiben Sie eine Datei "plot.cc", die einen um den Ursprung zentrierten `Canvas` der Breite 4 und Höhe 3 erzeugt, der 4000 Pixel in der Horizontalen und 3000 in der Vertikalen verwendet. Tragen Sie dann die Werte der folgenden Funktion als Grauwerte ein:

$$f(x, y) := \max(0, 100 \cdot (\sin(x^{-1}) \cdot \sin(y^{-1}) + 1))$$

Benutzen Sie die bereitgestellte Headerdatei `pgm.hh`, um die Pixel in eine Bilddatei zu schreiben, und prüfen Sie, dass das Ergebnis Ihren Erwartungen entspricht.

Fügen Sie hierzu der Klasse `Canvas` eine neue Methode `write(const std::string& filename)` hinzu.

Hinweise:

- Um das Pixel-Array zu initialisieren, können Sie folgenden Ausdruck verwenden (angenommen, die Variable mit dem Array heisst `_pixels`):

```
1 _pixels(horPixels,{{vertPixels}})
```

Dabei konstruiert die Anweisung mit den geschweiften Klammern einen Vektor der Länge `vertPixels`, der dann einmal für jede Spalte im Array (`horPixels` mal) kopiert wird.

- Die max-Funktion in der Formel oben soll sicherstellen, dass die Koordinatenachsen keine Auswertungsprobleme verursachen. Die Addition von 1 und Skalierung dienen dazu, Grauwerte im positiven Bereich mit genügend Abstufungen zu erzeugen. Das könnte man mit etwas zusätzlichem Aufwand auch automatisiert erledigen (siehe Fortgeschrittenen-Aufgabe).

(Grundlagen)

Übung 3 *Pixelgraphiken für Fortgeschrittene*

In dieser Aufgabe sollen Sie sich wie die anderen Teilnehmer mit der Erzeugung von Pixelgraphiken beschäftigen, allerdings mit ein paar Abänderungen und Ergänzungen. Lesen Sie dazu den Inhalt der Aufgabe "Pixelgraphiken", um mit der Aufgabenstellung vertraut zu sein. Programmieren Sie die dort beschriebene Funktionalität, allerdings mit folgenden Änderungen:

(a) Die Klasse `Canvas` soll intern `double` statt `int` zum Speichern der Grauwerte verwenden. Ein direkter Zugriff auf diese Werte soll nicht, und beim Abgreifen eines Pixelwerts über eine passende Methode `int pixel(int i, int j)` soll eine Umrechnung erfolgen.

(b) Diese Umrechnung soll dem kleinsten vorhandenen Grauwert den Wert 0 zuordnen, und dem größten Wert eine Zahl, die Ihrer Meinung nach genügend Abstufungen ermöglicht und mit dem Ausgabeformat verträglich ist (siehe unten). Dazwischen soll linear interpoliert werden.

(c) Statt die bereitgestellte Datei `pgm.hh` zu nutzen, erzeugt ihr Programm ein PNG. Informieren Sie sich dazu über die Nutzung von `libpng`. Eine mögliche Anlaufstelle, die den Umgang kompakt darstellt, ist zum Beispiel `http://zarb.org/~gc/html/libpng.html`. Benutzen Sie nicht direkt diese Vorlage, sondern kapseln Sie den nötigen C-Code mithilfe einer Klasse – je nach Geschmack können Sie dazu auch die Funktionalität von `canvas` um Methoden für das Lesen und Schreiben von PNGs ergänzen. Erzeugen Sie Grayscale-PNGs, falls Sie sich das zutrauen, oder alternativ RGB-PNGs, bei denen alle Channel den gleichen Wert haben.

(d) Testen Sie Ihr Programm mit der angegebenen Funktion $f(x, y)$, aber lassen Sie dabei die Addition von 1 und die Skalierung weg. Prüfen Sie, dass Ihr Programm trotzdem ein lesbares Bild

erzeugt, und stellen Sie sicher, dass das Lesen und direkte Schreiben eines Bildes dessen Inhalt nicht ändert.

Hinweis: Um die libpng-Bibliothek verwenden zu können, müssen Sie in der virtuellen Maschine oder auf Ihrem Laptop das Paket `libpng-devel` (unter Debian-Derivaten `libpng-headers`) installieren. In der virtuellen Maschine funktioniert das mit `sudo dnf install libpng-devel`.

(Fortgeschritten)