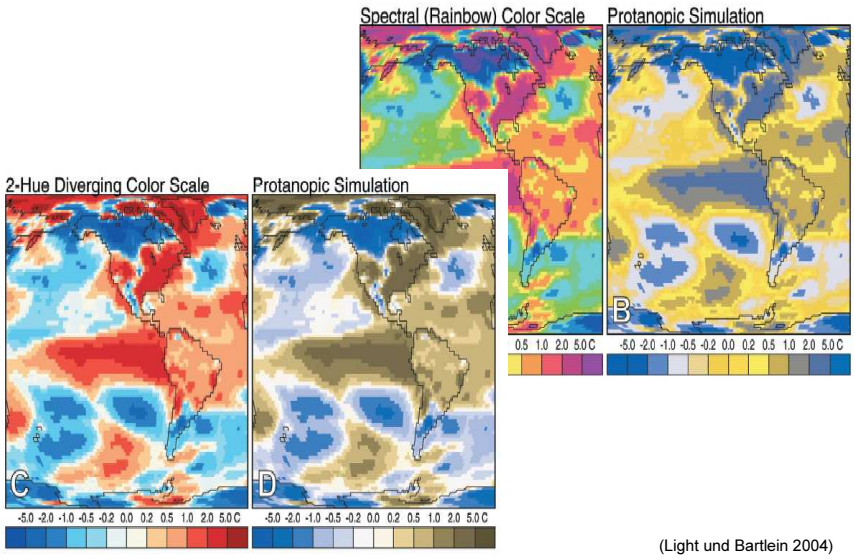
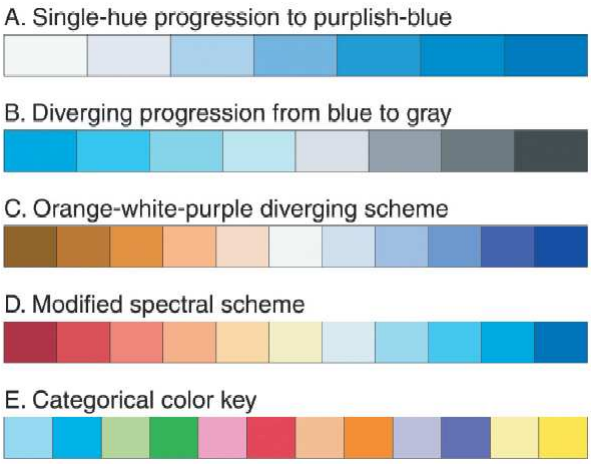
Nachtrag: Farben

Farbblindheit



(Light und Bartlein 2004)

Vorgeschlagene Farbskalen



(Light and Bartlein 2004)

Farbkodierung metrisch skalierter Daten

Unterscheide:

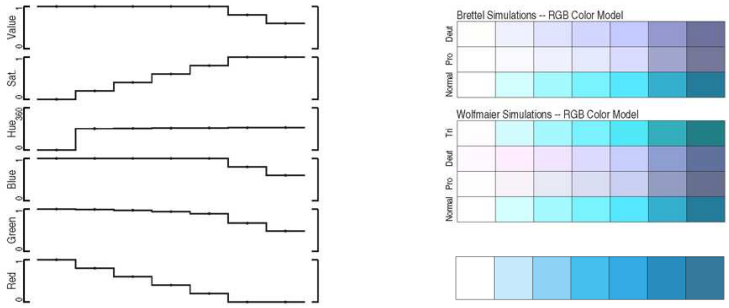
1. **Sequential Data** (ohne Betonung der Extrema): Kontinuierliche Zunahme des Farbwerts oder der Intensität

2. **Diverging Data** (Betonung der Abweichungen vom Mittelwert): Verwendung zweier komplementärer Farbschemata, die sich von einem gemeinsmaen Farbwert entfernen.

(http://geography.uoregon.edu/datagraphics/patterns/index.htm)

Farbkodierung metrisch skalierter Daten

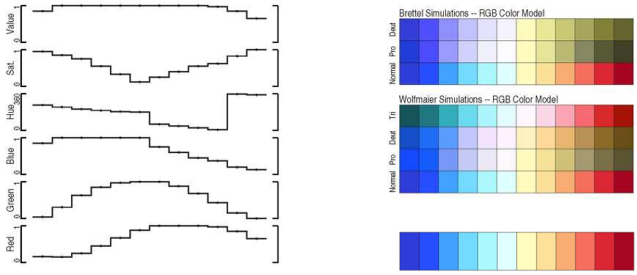
1. **Sequential Data** (ohne Betonung der Extrema):
   => Kontinuierliche Zunahme des Farbwerts oder der Intensität



(http://geography.uoregon.edu/datagraphics/patterns/index.htm)

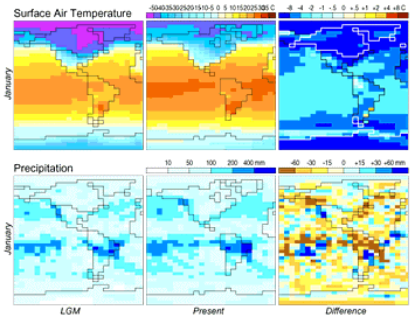
Farbkodierung metrisch skalierter Daten

2. **Diverging Data** (Betonung der Abweichungen vom Mittelwert):
   => Verwendung zweier komplementärer Farbschemata, die sich von einem gemeinsamen Farbwert entfernen



(http://geography.uoregon.edu/datagraphics/patterns/index.htm)

Darstellung von Differenzen



=> Darstellung von Differenzen in einer Karte ist oft geschickter als die Gegenüberstellung der beiden Karten.

(http://geography.uoregon.edu/datagraphics/patterns/index.htm)

Abfragesprachen

RA = Relationale Algebra

- Algebra = abgeschlossener und konsistenter Satz von Regeln
- Einfache Relationen können zu komplexen Relationen vereinigt werden
- formale Grundlage von Abfragesprachen, z.B. SQL
- bestehend aus **Operanden** und **Operatoren**

6 Typen von Operationen

1. **Select** (Selektion):	Auswahl bestimmter *Zeilen* (Tupeln) einer Tabelle
2. **Project** (Projektion):	Auswahl bestimmter *Spalten* (Attribute) einer Tabelle
3. **Union** (Vereinigung):	Auswahl *aller* Tupel, die in $n \geq 2$ Tabellen enthalten sind
4. **Intersection** (Schnittmenge):	Auswahl aller Tupel, die in *jeder* von $n \geq 2$ Tabellen enthalten sind
5. **Difference** (Differenz):	Auswahl aller Tupel einer Tabelle R, die *nicht* in Tabelle S enthalten sind
6. **Cross-Product** (Kreuzprodukt):	Vereinigung der Attribute aller $n \geq 2$ Tabellen; jedes Tupel von R mit jedem Tupel von S in Verbindung gebracht

Zusammengesetzte Operationen

Beispiele

- **Selection** + **Projection**: Auswahl einzelner Tupel mit ausgewählten Domänen („Selection“ i.w.S.)
- **Join** = **Cross-Product** + **Select**: Zusammenführung der Attribute aus verschiedenen Tabellen, anschl. Auswahl bestimmter Tupel
  - **Inner Join**: Auswahl aller Tupel einer Tabelle R, die auch in Tabelle S enthalten sind
  - **Outer Join**: Auswahl aller Tupel, die entweder in Tabelle R oder in Tabelle S oder in beiden enthalten sind

SQL

= "**S**tructured **Q**uery **L**anguage"

zusammengesetzt aus:

- **Data Definition Language** (**DDL**)
- **Data Modification Language** (**DML**)
- **Query Language (QL)**

SQL - Data Definition Language

Beispiele:

- CREATE TABLE city {name CHAR (50) NOT NULL, country CHAR (50), population NUMBER, PRIMARY KEY name};
- CHECK country IN ('Deutschland', 'Tschechien', 'Holland', 'Polen');
- GRANT SELECT ON city TO user_anton;

SQL - Data Modification Language

Beispiele:

- INSERT INTO city (name, country, population) VALUES ('Bayreuth', 'Deutschland', '80000');
- UPDATE city SET population = 10.000 WHERE name = 'Neudorf';
- DELETE FROM city WHERE name = 'Bayreuth';

SQL – Query Language

Abfragen: SELECT *column_name* FROM *relations* WHERE *tuple-constraint*

Abfrage aus ***einer*** Tabelle:

```
SELECT name, country, population
FROM city
WHERE population > 50000;
```

Abfragen über ***mehrere*** Tabellen:

```
SELECT city.name, city.country river.name
FROM city, river
WHERE city.population > 50000 AND river.name = Rhein
SORT BY city.population;
```

SQL - Query Language

Beispiele:

- SELECT name FROM city WHERE country = 'Deutschland' OR country = 'Tschechien';
- SELECT name FROM city WHERE country LIKE 'land';
- SELECT name FROM city WHERE population BETWEEN 10.000 AND 50.000 ORDER BY population;
- SELECT city.name, city.population, fluss.stadtname FROM city, fluss WHERE city.name = fluss.stadtname;
- CREATE VIEW flussstaedte AS SELECT city.name, city.population, fluss.stadtname FROM city, fluss WHERE city.name = fluss.stadtname;

Erweiterungen von SQL

SQL unterstützt nur einfache Datentypen (string, integer, real, ...), aber nicht *geometrische Objekte* (Punkte, Linien, Polygone)
=> Erweiterungen für GIS notwendig

Bsp.: **OGIS**

- definiert von einem Konsortium versch. Software-Firmen zur Standardisierung von GIS-Systemen (***OGC*** = ***O****pen* ***G****IS* ***C****onsortium*)
- Implementierung in C, Java, SQL, etc.
- für Objekt-Modelle
- für einfache SELECT, PROJECT, JOIN-Abfragen
- keine Unterstützung z.B. von Richtungsangaben (rechts, vorne, Norden, ...)

langfristiger Trend: Objekt-relationale SQL (z.B. SQL3/SQL99)

OGIS: Zusätzliche Operationen

TABLE 3.9: A Sample of Operations Listed in the OGIS Standard for SQL [OGIS, 1999]

Basic Functions	`SpatialReference()`	Returns the underlying coordinate system of the geometry
	`Envelope()`	Returns the minimum orthogonal bounding rectangle of the geometry
	`Export()`	Returns the geometry in a different representation
	`IsEmpty()`	Returns true if the geometry is a null set
	`IsSimple()`	Returns true if the geometry is simple (no self-intersection)
	`Boundary()`	Returns the boundary of the geometry
Topological/ Set Operators	`Equal`	Returns true if the interior and boundary of the two geometries are spatially equal
	`Disjoint`	Returns true if the boundaries and interior do not intersect
	`Intersect`	Returns true if the geometries are not disjoint
	`Touch`	Returns true if the boundaries of two surfaces intersect but the interiors do not
	`Cross`	Returns true if the interior of a surface intersects with a curve
	`Within`	Returns true if the interior of the given geometry does not intersect with the exterior of another geometry
	`Contains`	Tests if the given geometry contains another given geometry
	`Overlap`	Returns true if the interiors of two geometries have nonempty intersection
Spatial Analysis	`Distance`	Returns the shortest distance between two geometries
	`Buffer`	Returns a geometry that consists of all points whose distance from the given geometry is less than or equal to the specified distance
	`ConvexHull`	Returns the smallest convex geometric set enclosing the geometry
	`Intersection`	Returns the geometric intersection of two geometries
	`Union`	Returns the geometric union of two geometries
	`Difference`	Returns the portion of a geometry that does not intersect with another given geometry
	`SymmDiff`	Returns the portions of two geometries that do not intersect with each other

(Shekhar und Chawla 2003)